

REPRESENTATION AND IDENTIFICATION
OF ALPHA-NUMERIC CHARACTERS
USING GALOIS TRANSFORM

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

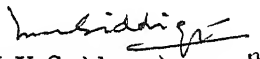
by
ALOK GUPTA

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

November, 1992

CERTIFICATE

It is certified that the work contained in the thesis entitled "REPRESENTATION AND IDENTIFICATION OF ALPHA-NUMERIC CHARACTERS USING GALOIS TRANSFORM" has been carried out by Alok Gupta under my supervision and that this work has not been submitted elsewhere for a degree


(M U Siddiqui) 30.11.92

Professor

Department of Electrical Engineering

I I T Kanpur

November, 1992

08 APR 1993

EE

CENTRAL LIBRARY
IIT KANPUR

Acc. No. A. 115435

EE-1993-M-GUP-REP

ACKNOWLEDGEMENT

I am deeply grateful to my thesis supervisor Prof M U Siddiqui for suggesting me this topic and his valuable guidance throughout the work. He has been a constant source of inspiration and motivation to me.

My special thanks to Galgali and Subbarao for their suggestions and cooperation during my early phase of work. I would also like to thank Madhu, Uday, Balwinder and Venkatesh for their help during various stages of my thesis work.

Working in Lab was made more pleasant due to the company of Pandey, Anurag, Mahajan, Ravi and Nandini. My stay at IIT was made a memorable one by C-TOP, Hemant, Kholu, Anil, Mohalik and Vijay.

DEDICATED

TO

MY PARENTS

ABSTRACT

We obtain two dimensional Galois transform representation of different image sizes. Calculation of Galois transform coefficients has been carried out by using FFT, systolic array and Horner's rule. Four T414 Transputers and five T800 Transputers have been used in linear and mesh configurations for this purpose. Various algorithms are evaluated by comparing sequential, parallel one node and parallel 8 node timings for different image sizes. As a concrete application, FFT based configuration with mesh connected processors is used to calculate Galois coefficients for IBM font of English alphabet and Arabic numerals. The coefficients are used to identify these characters by comparing them with coefficients stored in computer memory. Two algorithms have been implemented for this purpose. Their performance is compared to that of a straight sample domain algorithm in terms of the average number of comparisons required for identification in each case.

TABLE OF CONTENTS

Chapter 1	INTRODUCTION	1
1 1	Motivation	1
1 2	Scope of the work	2
1 3	Organization of the Thesis	3
Chapter 2	THEORY OF GALOIS SWITCHING FUNCTIONS	4
2 1	1-D Galois switching function(GSF)	4
2 2	2-D Galois switching function	7
2 2 1	Algebraic relations	7
2 2 2	Conjugacy relations	8
2 2 3	Number of frobenius cycles	8
Chapter 3	TRANSPUTER AND OCCAM	11
3 1	Transputer	11
3 1 1	Introduction	11
3 1 2	Architectural details	12
3 1 2 1	Instruction processor	13
3 1 2 2	Instruction set	13
3 1 2 3	Memory controller	14
3 1 2 4	Process scheduler	14
3 1 2 5	Communication	16
3 1 2 6	Communication links	17
3 1 2 7	Timer	17

3 1 2 8	Floating point processor	18
3 2	Occam	18
3 3	Programming Transputers	20
3 3 1	Topology	20
3 3 2	Placement	20
3 3 3	Non determinacy	21
3 3 4	Deadlock	22
3 3 5	General algorithm design consideration	22
3 4	Setup of a transputer network system	25
3 4 1	Hardware setup of the transputer network system	26
3 4 2	Code development	28
Chapter 4	ALGORITHMS FOR CALCULATING COEFFICIENTS	29
4 1	Using systolic array	29
4 1 1	Design methodology for systolic arrays	30
4 1 1 1	Systolic array design for matrix multiplication	30
4 1 2	Algorithm for systolic array implementation	32
4 2	Horner's rule implementation	33
4 2 1	Algorithm for polynomial computation	34
4 3	Fast Fourier transform	35
4 3 1	Cooley Tukey algorithm	35
4 3 2	Algorithm for FFT computation	37

Chapter 5	IDENTIFICATION OF ENGLISH	
ALPHABET AND ARABIC NUMERALS		39
5 1	Introduction	39
5 2	Algebraic properties of Galois coefficients	41
5 3	Method 1 Identification in coefficient space	44
5 4	Method 2 Identification in coefficient space	46
5 5	Method 3 Identification in Real space	48
5 6	Performance comparison	49
Chapter 6	SUMMARY AND CONCLUSIONS	51
6 1	Results	51
6 2	Scope for further Work	54
APPENDIX A	8×8 binary pixel representation of English	
alphabet and Arabic numerals		56
APPENDIX B	Galois coefficients of English alphabet and	
Arabic numerals		60
REFERENCES		64

CHAPTER ONE

INTRODUCTION

1.1 MOTIVATION

Man has been living on earth since time—immemorial. He had to fight for his survival. In order to survive, he needed to identify his enemies and friends quickly. Therefore, he has developed the ability to identify and distinguish between things instantaneously. However, his survival was not dependent on his ability to deal with large numbers and therefore his ability in that field is limited. Given two 100 digit numbers, man would take hours to multiply them whereas a simple microprocessor can do it in micro seconds. On the other hand, a microprocessor/computer has a very limited ability to identify things.

With advances in science and technology, automation has replaced manual effort. Computers encompass a wide range of subjects including image—processing. The problems in the field of image—processing which draw our attention are

- (1) Can computers be used for identification purposes?
- (2) Can a method/methods be devised to be used for differentiating between two images which are almost similar?
- (3) Is the above method dependent upon the type of image?
- (4) Can this method operate in real time conditions?
- (5) What are the applications in which this method can be used?
- (6) Whether the method is commercially viable?

1 2 SCOPE OF THE WORK

In this thesis we will try to tackle some of the the above questions Further, we will also try to devise some methods for identifying the English language alphabet and Arabic numerals As the number of methods that can be devised is very large, we have restricted our attention to the methods which use Galois coefficients

The method used by the human brain for identifying images is based on the pattern matching of an image with images stored in its memory This process appears so natural (i e without any physical labor for human being) that we do not comprehend the vast neural network required for such quick matching Some advances have been made in fabricating large neural networks in achieving pattern matching but the huge storage requirements inhibit the use of this technology in the present day conditions In this thesis we will try to devise a process that can achieve the purpose in real time conditions Methods to achieve the above purpose on the computer should bear in mind, that it should be based on some intrinsic property/transform of image which is computer friendly

Here computer friendly property/transform of a image means that the method using property/transform is specially suited for computer in the same way as pattern matching is suited for human brain As computers are basically large number crunchers, hence the method developed can afford to be calculation intensive

One of the property which is intrinsic to a image is the Galois transform of the image (as it is a one to one transform) We have to see whether Galois transform of image can be used for identifying the image Since Galois transform of an image is computation intensive, we will be using parallel processing for its real time implementation We will then use the above transform for representation and identification of English language alphabets and Arabic numerals

1 3 ORGANIZATION OF THE WORK

Chapter 2 gives necessary algebraic background for calculation of Galois transform in two variable case. It also discusses the possible conjugacy constraints and conjugacy classes.

In Chapter 3, we discuss the transputer and occam facility that we have used for developing parallel algorithms. We also discuss the hardware setup of the transputer network system and General algorithm design considerations.

In Chapter 4, we develop the algorithms for calculation of coefficients. They are based on fft(linear arrangement of processors), fft (processors in mesh arrangement), Horner's rule and systolic array.

In Chapter 5, we discuss the algebraic properties of Galois coefficients for 8×8 binary images of alphabet. Also, the three methods that have been developed for identification are discussed and compared. We also evaluate the advantages and drawbacks of each method over one another.

In chapter 6, we compare the sequential, parallel—one-node and parallel—eight-node timings in the calculation of coefficients for different algorithms. We conclude it with the discussion on the scope for future work.

CHAPTER 2

THEORY OF GALOIS SWITCHING FUNCTIONS

A combinatorial network with k inputs and n outputs may be represented by a set of n switching functions of k variables over $GF(2)$. By employing *finite fields*, it is possible to represent a set of n functions of k variables by a single variable polynomial over an appropriate extension field of $GF(2)$. The functions described by these polynomials which essentially represent mapping from $GF(2^k)$ to $GF(2^n)$ are called 1 dimensional Galois switching functions. The coefficients of this polynomial (termed as Galois polynomial) are called Galois coefficients. These polynomials have a well defined algebraic structure and possess remarkable properties based on Frobenius cycles. The concept of 1-D GSFs described by single variable may be extended to multi dimensional GSFs, especially two dimensional GSFs described by two variable GPs. The 2-D GSFs are particularly useful for representation and processing of images (pictorial data) represented in the form of two variable GPs. We will first confine our attention to 1-D GSFs and then study in detail the properties of 2-D GSFs.

2.1 1-D GALOIS SWITCHING FUNCTION (GSF) [5]

Any non-zero element of a finite field can be represented by a power α^j (polar form) of a primitive element α and at the same time can be represented by a polynomial of α

$$\zeta_{k-1}\alpha^{k-1} + \zeta_{k-2}\alpha^{k-2} + \dots + \zeta_1\alpha + \zeta_0, \quad \zeta_i \in GF(2)$$

if the minimal polynomial over $GF(2)$ of α is given. Thus we can take the set of coefficients $\zeta_{k-1}, \zeta_{k-2}, \dots, \zeta_0$ as cartesian representation of finite field elements. For example let α be

the primitive element of $GF(2^3)$ and x^3+x^2+1 be the minimal polynomial over $GF(2)$ of α then truth table can be written as

polar form	cartesian form		
X	x_2	x_1	x_0
0	0	0	0
1	0	0	1
α	0	1	0
α^2	1	0	0
α^3	1	0	1
α^4	1	1	1
α^5	0	1	1
α^6	1	1	0

A polynomial expression for a 1-D GSF is to be obtained, in which the value of the polynomial $f(x)$ at $x = \alpha^j$ is equal to the value of function at index α^j . This $f(x)$ can be written as

$$f(x) = a_{\alpha^{-\infty}} + a_{\alpha^0}x^{2^k-1} + \sum_{i=1}^{2^k-2} a_{\alpha^i}x^{2^k-i-1} \quad (2.1.1)$$

The coefficient vector

$$a = (a_{\alpha^{-\infty}}, a_{\alpha^0}, a_{\alpha^1}, \dots, a_{\alpha^{\zeta}}) \quad \text{where} \quad \zeta = 2^k - 2$$

and the function value vector

$$f = (f_{\alpha^{-\infty}}, f_{\alpha^0}, f_{\alpha^1}, \dots, f_{\alpha^{\zeta}}) \quad \text{where} \quad \zeta = 2^k - 2$$

can be related through nonsingular $2^k \times 2^k$ matrices given below

$$\begin{bmatrix} f(0) \\ f(1) \\ f(\alpha) \\ f(\alpha^2) \\ \\ \\ f(\alpha^s) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & (\alpha^s) & & \alpha^2 & \alpha \\ 1 & 1 & (\alpha^s)^2 & & \alpha^4 & \alpha^2 \\ 1 & 1 & & & & \\ 1 & 1 & & & & \\ 1 & 1 & \alpha & & \alpha^{s-1} & \alpha^s \end{bmatrix} \begin{bmatrix} a_{\alpha^{-\infty}} \\ a_{\alpha^0} \\ a_{\alpha} \\ a_{\alpha^2} \\ \\ \\ a_{\alpha^s} \end{bmatrix}$$

$$\begin{bmatrix} a_{\alpha^{-\infty}} \\ a_{\alpha^0} \\ a_{\alpha} \\ a_{\alpha^2} \\ \\ \\ a_{\alpha^s} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 \\ 1 & 1 & 1 & & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & & \alpha^s \\ 0 & 1 & 1 & \alpha^2 & (\alpha^2)^s & \\ 0 & 1 & & & & \\ 0 & 1 & & & & \\ 0 & 1 & \alpha^s & & \alpha^2 & \alpha \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(\alpha) \\ f(\alpha^2) \\ \\ \\ f(\alpha^s) \end{bmatrix}$$

One more interesting property of Galois coefficients can be seen by observing the above matrix. It can be broken into four parts

$$\left[\begin{array}{c|cc} 1 & 0 & 0 \\ \hline 0 & \mathbf{H} & \end{array} \right]$$

This H matrix is a DFT matrix of order $2^k - 1$ over an appropriate extension field of GF(2), therefore coefficients and DFT of function are related. The relations are given below

$$a_{\alpha^{-\infty}} = F_{\alpha^{-\infty}}, \quad a_{\alpha^0} = (F_{\alpha^{-\infty}} + F_{\alpha^0}) \quad \text{and} \quad a_{\alpha^1} = F_{\alpha^1}$$

2 2 TWO DIMENSIONAL GSFs

2 2 1 ALGEBRAIC RELATIONS [5]

Any 2-D GSF $f(x_1, x_2)$ can be represented by a 2 variable GP

$$f(x_1, x_2) = \sum_{j_1} \sum_{j_2} a_{j_1 j_2} x_1^{-j_1} x_2^{-j_2} \quad (2\ 2\ 1\ 1)$$

where $j_1 = -\infty, 0, 1, 2, \dots, 2^{k_1}-2$ and

$$j_2 = -\infty, 0, 1, 2, \dots, 2^{k_2}-2$$

The coefficients are given by

$$a_{-\infty-\infty} = f(\alpha_1^{-\infty}, \alpha_2^{-\infty}) \quad (2\ 2\ 1\ 2\ a)$$

$$a_{j_1-\infty} = \sum_{x_1} x_1^{j_1} f(x_1, \alpha_2^{-\infty}) \quad (2\ 2\ 1\ 2\ b)$$

$$a_{-\infty j_2} = \sum_{x_2} x_2^{j_2} f(\alpha_1^{-\infty}, x_2) \quad (2\ 2\ 1\ 2\ c)$$

$$a_{j_1 j_2} = \sum_{x_1} \sum_{x_2} x_1^{j_1} x_2^{j_2} f(x_1, x_2) \quad (2\ 2\ 1\ 2\ d)$$

where

$$j_1 = -\infty, 0, 1, 2, \dots, 2^{k_1}-2 \text{ and}$$

$$j_2 = -\infty, 0, 1, 2, \dots, 2^{k_2}-2$$

α_1 is a primitive element of $GF(2^{k_1})$, coefficients (a's) $\in GF(2^L)$, L being the L C M of n, k_1 and k_2

If we arrange the function values $f(x_1, x_2)$ in the form of a $2^{k_1} \times 2^{k_2}$ matrix, then the coefficients of the 2 variable GP can be obtained by

(1) Computing the 1-D Galois transform coefficients of the rows of the matrix, which represents a mapping from $GF(2^{k_1})$ to $GF(2^n)$, and replacing the rows with the resulting coefficients which belong to say, $GF(2^{L_1})$, L_1 being the L C M of k_1 and n , followed by

(2) Computing the 1-D GT coefficients of the resulting columns, which now represent a mapping from $GF(2^{k_2})$ to $GF(2^{L_1})$. The resulting final coefficients belong to $GF(2^L)$, L being the L C M of k_2 and L_1

2 2 2 CONJUGACY RELATIONS [5]

If at least one k_1 does not divide n , the coefficients $a_{j_1 j_2}$ satisfy the conjugacy relations given by

$$(a_{-\omega_{j_2}})^Q = a_{-\omega_{j_2} Q \pmod{2^{k_2}-1}} \quad (2\ 2\ 2\ 1\ a)$$

$$(a_{j_1 j_2})^Q = a_{j_1 Q \pmod{2^{k_1}-1} j_2 Q \pmod{2^{k_2}-1}} \quad (2\ 2\ 2\ 1\ b)$$

where $j_1 = -\omega, 0, 1, 2, \dots, 2^{k_1}-2$ and $Q=2^n$

If both k_1 divide n then fields to which the function values and the coefficients belong is same if

(1) the function values belong to $GF(2^n)$, and not to any of its sub fields

(2) all the function values belong to $GF(2^n)$ as well as to a subfield of it,

namely $GF(2^{n_1})$, and if both k_1 divides n_1 also

Thus in cases (1) and (2), the GP coefficients exhibit trivial conjugacy relations and they belong to $GF(2^n)$ and $GF(2^{n_1})$ respectively

(3) If all the function values belong to $GF(2^n)$ as well as to a subfield of it, namely $GF(2^{n_1})$, and if atleast one k_1 does not divides n_1 then conjugacy relation exists (given by 2 2 2 1 with $Q=2^{n_1}$) and coefficients belong to $GF(2^L)$ where $L=L\ C\ M$ of n_1, k_1 and k_2

2 2 3 NUMBER OF FROBENIUS CYCLES [5]

Frobenius cycles exist only if $GF(2^n)$ is a proper subfield of $GF(2^L)$

The number of frobenius cycles in the case of two variable GPs, is given by

$$\text{nfrob} = 1 + \sum_{\substack{D_1 | M_1 \\ D_1 \neq 1}} \phi(D_1) / \exp_Q(D_1) + \sum_{\substack{D_2 | M_2 \\ D_2 \neq 1}} \phi(D_2) / \exp_Q(D_2) + \\ \sum_{\substack{D_1 | M_1 \\ D_1 \neq 1}} \sum_{\substack{D_2 | M_2 \\ D_2 \neq 1}} \phi(D_1) \phi(D_2) / \text{L C M}(\exp_Q(D_1), \exp_Q(D_2)) \quad (2 \ 2 \ 3 \ 1)$$

where $M_1 = 2^{k_1} - 1$, $M_2 = 2^{k_2} - 1$, $Q = 2^n$, $\exp_Q(D_1) = e$ is the least positive integer such that $Q^e \equiv 1 \pmod{D_1}$ and $\phi(D_1)$ is the Euler's phi function (the number of integers smaller than D_1 , which are relatively prime to D_1)

Example Let $n = 1, k_1 = 3, k_2 = 3$

Then $M_1 = M_2 = 7$, $Q = 2$,

The divisors of 7 are 1 and 7,

$\exp_2(1) = 1$ and $\exp_2(7) = 3$,

$\phi(1) = 1$ and $\phi(7) = 6$,

Using (2 2 3 1)

Thus $\text{nfrob} = 1 + 1/1 + 6/3 + 1/1 + 6/3 + 1 \times 1 / \text{L C M of } (1,1) + 1 \times 6 / \text{L C M of } (1,3) + 6 \times 1 / \text{L C M of } (3,1) + 6 \times 6 / \text{L C M of } (3,3)$

$$\text{nfrob} = 1 + 1 + 2 + 1 + 2 + 1 + 2 + 2 + 12$$

$$\text{nfrob} = 24$$

conjugacy relation in this case would be

$$(a_{j_1 j_2})^2 = a_{2 j_1 (\text{mod } 7) \ 2 j_2 (\text{mod } 7)},$$

$$j_1 = -\infty, 0, 1, \dots, 6, j_2 = -\infty, 0, 1, \dots, 6,$$

Now we list the conjugacy classes

- | | | | |
|--|--|------------------------|------------------|
| (1) $\{(-\infty, -\infty)\}$ | (2) $\{(-\infty, 0)\}$ | (3) $\{(0, -\infty)\}$ | (4) $\{(0, 0)\}$ |
| (5) $\{(-\infty, 1), (-\infty, 2), (-\infty, 4)\}$ | (6) $\{(-\infty, 3), (-\infty, 6), (-\infty, 5)\}$ | | |
| (7) $\{(0, 1), (0, 2), (0, 4)\}$ | (8) $\{(0, 3), (0, 6), (0, 5)\}$ | | |
| (9) $\{(1, -\infty), (2, -\infty), (4, -\infty)\}$ | (10) $\{(1, 0), (2, 0), (4, 0)\}$ | | |

$$(11) \{(1,1), (2,2), (4,4)\}$$

$$(13) \{(1,3), (2,6), (4,5)\}$$

$$(15) \{(1,5), (2,3), (4,6)\}$$

$$(17) \{(3, \infty), (6, \infty), (5, \infty)\}$$

$$(19) \{(3,1), (6,2), (5,4)\}$$

$$(21) \{(3,3), (6,6), (5,5)\}$$

$$(23) \{(3,5), (6,3), (5,6)\}$$

$$(12) \{(1,2), (2,4), (4,1)\}$$

$$(14) \{(1,4), (2,1), (4,2)\}$$

$$(16) \{(1,6), (2,5), (4,3)\}$$

$$(18) \{(3,0), (6,0), (5,0)\}$$

$$(20) \{(3,2), (6,4), (5,1)\}$$

$$(22) \{(3,4), (6,1), (5,2)\}$$

$$(24) \{(3,6), (6,5), (5,3)\}$$

We will be using these conjugacy constraints in the calculation of Galois coefficients of alphabets and numerals of size 8×8

CHAPTER 3

TRANSPUTER AND OCCAM

3.1 TRANSPUTER [2]

3.1.1 INTRODUCTION

The word *Transputer* may be interpreted as a contraction of the words *Tranceiver* and *computer*. The interpretation suggests that the Transputer consists of a communication system and a computational element. "Transputer" can be used as a basic element in multiprocessor systems. Transputer can be distinguished from any other processor due to a number of built-in features, like high processor speed, low chip count, and simplicity of the system design. The chip, being very versatile, has received considerable attention and popularity among concurrent system designers.

A Transputer can be used in a single processor system or in networks to build high performance concurrent systems. A typical member of transputer product family is a single chip containing processor, memory and point-to-point communication links. A network of Transputers can be easily constructed using these links. As a microcomputer, the Transputer is unusual in its ability to communicate with other Transputers. A variety of different configurations can be built by hard-wiring Transputers together, with no separate switching and forwarding network, limited only by the number of links provided on each Transputer. The current Transputer systems have four to six links. Four links are enough to allow enormous range of useful configurations.

The major advantages of Transputer based system are

Scalability It is much easier to enhance system by adding further transputer to it than would be the case with other microprocessors.

Compatibility — the ease of replacing one transputer model with another without major design changes in a system. This extends to mixing models of transputer within one system.

3 1 2 ARCHITECTURAL DETAILS

Transputers are often described as RISC processors. The computational instructions follow RISC principles closely, and they attain the benefits claimed for RISC architecture. However, they also have a small number of important non-RISC instructions concerned with scheduling and message passing.

The Transputer is unusual in its ability to execute many software processes at the same time. A program can be run on a single Transputer, in which case the concurrency of the processes will be *simulated* by hardware with no software intervention. Provided the communication between the subprocesses is not too complicated, the same program can also be distributed over several processors (transputers), in which case the component processes will be run in real concurrency. Just as in a single processor, interprocess message passing and the necessary synchronization (between directly connected Transputers) are achieved in hardware, and no operating system is needed.

The inbuilt features of the Transputer are

- Instruction processor

- Small amount of on chip memory

- Memory controller

- DMA control for four independent fast links

- A microcoded multitasking kernel

- And an elapsed-time clock

The best known Transputers are the T212, T414 and T800. The T212 is a 16-bit processor, the other two are 32-bit processors. The T800 has a full IEEE floating-point processor on chip.

3 1 2 1 INSTRUCTION PROCESSOR

The processor portion of a Transputer is a traditional microprocessor. The processor normally obtains its instructions and data from the internal 4K RAM. Data and instructions can also be obtained from the links. The processor provides 32-bit addressing. Memory is addressed byte-wise and stored in 4-byte units. Software sees no difference between on-chip and external memory except in speed.

The design of Transputer processor exploits the availability of fast on-chip memory by having only a small number of registers; the CPU contains six registers which are used in the execution of a sequential process. The small number of registers, together with the simplicity of the instruction set, enables the processor to have relatively simple (and fast) data-paths and control logic.

3 1 2 2 INSTRUCTION SET

As in RISC architecture, transputer instruction set is designed for simple and efficient compilation. All the instructions have the same format and chosen to give a compact representation of the operations which are most frequently occurring in programs. The instruction size of the Transputer is 8 bits for most instructions. There are prefix instructions which allow the operand to be extended to any length. Measurements show that about 70% of the executed instructions are encoded in a single byte. Short instructions improve the effectiveness of the instruction prefetch, which in turn improves processor performance. There is an extra word of prefetch buffer, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is short, there is little time penalty when a jump instruction causes the buffer contents to be discarded. Also, the instruction set is independent of processor word length, allowing the same microcode to be used for Transputers with different word lengths.

3 1 2 3 MEMORY CONTROLLER

The memory controller can drive external dynamic RAM with no additional circuitry. Together with the controller, the processor can address a linear address space of 4-Gbytes. The 32-bit wide memory interface uses multiplexed data and address lines and provides a data rate of up to 4 bytes every 100 nanoseconds (40 Mbytes/sec) for a 30 MHz device. The configurable memory controller provides all timing, control and DRAM refresh signals for a wide variety of mixed memory systems.

3 1 2 4 PROCESS SCHEDULER

The transputer provides efficient support for concurrency and communication. It has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel. The processor does not need to support the dynamic allocation of the storage as the compiler is able to perform the allocation of space to the concurrent processes.

A process starts, performs a number of actions, and then terminates. Typically a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority. At any time, a concurrent process may be

- | | | |
|----------|---|----------------------------------|
| active | — | being executed |
| | — | on a list waiting to be executed |
| inactive | — | ready to input |
| | — | ready to output |
| | — | waiting until a specified time |

The scheduler operates in such a way that inactive processes do not consume any processor time. The active processes waiting to be executed are held on a list. This is a linked list of process workspaces, implemented using two registers, one of which points to the first process on the list, the other to the last.

The IMS T800 supports two levels of priority. Priority 1 (low priority) processes are executed whenever there are no active priority 0 (high priority) processes. High priority processes are expected to execute for a short time.

If one or more high priority processes are able to proceed, then one is selected and run until it has to wait for communication, a timer input, or until it completes processing. If no process at high priority is able to proceed, but one or more processes at low priority are able to proceed, then one is selected. Each process runs until it has completed its action, but is descheduled whilst waiting for communication from another process or Transputer, or for a time delay to complete. In order for several processes to operate in parallel, a low priority process is only permitted to run for a maximum of two time slices before it is forcibly descheduled at the next descheduling point. The time slice period is 5120 cycles of the external 5 MHz clock, giving ticks approximately 1ms apart. A process can only be descheduled on certain instructions, known as descheduling points. As a result, an expression evaluation can be guaranteed to execute without the process being time sliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the processor workspace and the next processor is taken from the list. Process scheduling pointers are updated by instructions which cause scheduling operations and should not be altered directly. Actual process switch times are less than 1 μ s, as little state needs to be saved and it is not necessary to save the evaluation stack on rescheduling.

The processor provides a number of special operations to support the processor model, including start process and end process. When a main process executes a parallel construct, start process instructions are used to create the necessary additional concurrent processes. A start process instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the end of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by the use of the end process instruction. This uses a workspace location as a counter of the parallel construct components which have still to terminate. The counter is initialized to the number of components before the processes are started. Each component ends with an end process instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

3 1 2 5 COMMUNICATIONS

Communication between the processes is achieved by means of channels. The communication is point-to-point, synchronized and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer. A channel between two processes executing on the same Transputer is implemented by a single word in memory, and a channel between processes executing on different Transputers is implemented by point-to-point links. The processor provides a number of operations to support message passing, the most important being *input message* and *output message*.

The input message and output message instructions use the address of the channel to determine whether the channel is internal or external. This means that the same instruction sequence can be used for both for hard and soft channels, allowing a process to be written and compiled without the knowledge of where its channels are connected. The communication takes place when both the inputting and outputting processes are ready (Synchronicity). Consequently, the process which first becomes ready must wait until the second one is also ready. A process performs an input or output by loading the evaluation stack with a pointer to a message, the address of a channel, and a count of the number of bytes to be transferred, and then executing an input message or an output message instruction.

3 1 2 6 COMMUNICATION LINKS

A link between two transputers is implemented by connecting a link interface on one Transputer to a link interface on the other Transputer by two one-directional signal wires, along which data is transmitted serially. The two wires provide two channels, one in each direction. This requires a simple protocol to multiplex data and control information. Messages are transmitted as a sequence of bytes, each of which are to be acknowledged before the next is transmitted. A byte of data is transmitted as a start bit followed by a one bit followed by eight bits of data followed by a stop bit. An acknowledgement is transmitted as a start bit followed by a stop bit. An acknowledgement indicates that a process was able to receive the data byte and that it is able to buffer another byte.

The protocol permits an acknowledgement to be generated as soon as the receiver has identified a data packet. In this way the acknowledgement can be received by the transmitter before all of the data packets have been transmitted and the transmitter can transmit the next data packet immediately. The IMS T414 transputer does not implement this overlapping and achieves a data rate of 0.8 Mbytes per second using a link to transfer in one direction. However, by implementing sufficient overlapping and including sufficient buffering in the link hardware, the IMS T800 more than doubles this data rate to 8 Mbytes per second in one direction, and achieves 2.4 Mbytes per second when the link carries data in both directions.

3 1 2 7 TIMER

The Transputers have two timer clocks which 'tick' periodically. The timer provide accurate process timing, allowing processes to be descheduled until a specific time. In the IMS T800 Transputer, one timer is accessible to only high priority processes and is incremented every microsecond, cycling completely in approximately 4295 milliseconds. The other is accessible only to a low priority process and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of

approximately 76 hours

3 1 2 8 THE FLOATING POINT PROCESSOR

The IMS T800 has a full IEEE floating point processor on chip. The FPU operates concurrently with CPU. This means that it is possible to do address calculation in the CPU whilst the FPU performs the floating point calculation. This can lead to significant performance improvements in real applications which access arrays heavily. Performance depends on many things, including clock and memory speeds — for the 20 MHz T800 these figures are of the order of 10 RISC MIPS and 1 MFLOPS (these are not upper bounds).

As can be noticed from above, all the links can be active at the same time as well as the processor. Thus the Transputer can support nine truly concurrent activities (one link can transfer data in both directions, of course, memory accesses have to be interleaved). On a T800, the floating point processor operates in parallel with the instruction processor, which gives a tenth level of concurrency at the hardware level (but both the processors are controlled by a single instruction stream).

3 2 THE OCCAM

Entities are not to be multiplied beyond necessity was the philosophy of the original implementation of Occam. Occam is the native language for Transputers. The design of Occam has been heavily influenced by the work on *Communicating Sequential Processes* (CSP), which gives a mathematical framework for specifying the behaviour of parallel processes. Occam is based on the CSP model of computation, but with features chosen to ensure efficiency of implementation. In this model, an application is decomposed into a collection of communicating processes, and the processes communicate by passing messages.

Occam model is based on the idea of *process*. The software building block is a *process*. A system is designed in terms of an interconnected set of processes. Each *process*

can be regarded as an independent unit of design. Its internal design is hidden and is completely specified by the message it sends and receives. Internally, each process can be designed as a set of communicating processes. The system design is therefore hierarchically structured. Occam processes do not share any variables, nor semaphores. Occam does not require (nor support) shared memory. Messages pass from exactly one process to the other. There are no multiple senders or receivers, no broadcasting, and no uncertainty about where a message came from or where it is going. Messages are unbuffered, so sending and receiving a message involves momentary synchronization between the two participating processes. Messages are sent through static channels, as if through a circuit switched (rather than packet switched) network.

To gain the most benefit from the Transputer architecture, the whole system can be programmed in Occam. This provides all the advantages of a high level language, the maximum program efficiency and the ability to use the special features of the Transputers. The Occam model of concurrency is applicable equally to processes running on separate processors and to processes running within a single processor. Since the processor can be controlled only by one instruction stream, it is evident that *the processes in one processor cannot be truly concurrent*. However, the processes can be multiprogrammed, just as on a mainframe, so that the effect of concurrency is reproduced (apart from speed). This 'simulated concurrency' as distinct from 'real concurrency' is known as '*gratuitous concurrency*'. Within a Transputer, this multiprogramming is handled by hardware with no need for any operating system.

Occam provides a framework for designing concurrent systems using Transputers just in the same way that boolean algebra provides a framework for designing electronic systems from logic gates. The system designer's task is eased because of the architectural relationship between Occam and Transputer. A program running in a Transputer is formally equivalent to an Occam process, and so a network of Transputers can be described directly as an Occam program.

Occam, when compiled for execution on a Transputer, is ideal for embedded multiprocessor systems. Where it is required to exploit concurrency, but still to use standard languages, Occam can be used as a harness to link modules written in selected languages. Performance approaching that of assembled machine code can be achieved, by both matching of an architecture to a specific language, and the use of static memory allocation avoiding run-time memory range checking.

3 3 PROGRAMMING TRANSPUTERS [11]

The following are the points that concern the programmer of a concurrent system

3 3 1 TOPOLOGY

The pattern in which the processors are connected together is known as topology or the configuration. The idea of configuration depends on the assumption that processors are connected permanently, or at least for the life of a whole program, an assumption which is broadly true for current Transputer systems.

It is the responsibility of the designer of the overall system to decide how to configure the processors within it. Designing a topology for a large system can be difficult. It can sometimes be guided by an obvious mapping of a problem (or a solution) onto separate processors, but the designer is not always so fortunate and most often analyse several difficult possibilities using what help is obtainable from concurrency theory, graph theory, queuing theory etc. There is a strong tendency to fall back on standard, straight forward and well understood topologies even though they may be far from optimal.

3 3 2 PLACEMENT

Describing systems in terms of Occam processes allows algorithmic issues to be separated from the question of what hardware is going to perform those activities. This is a useful abstraction. One would like to be able to express an algorithm in the form of a program which is independent of hardware, so that it could be subsequently be performed

using many different networks of processors. Each implementation would need a specification of how many processors were needed, how they were to be connected, and which processes were to be installed on which processors, but the specification ought not to need any change in the program. The specification is called placement.

In practice, Transputer programs are not completely independent of their placement. Unless it is carefully designed, a program will only run on one particular network of Transputers, or on a small number of similar networks. To run on other networks, the program itself will have to be changed. Sometimes the changes will be minimal, but other cases may need extensive modifications. Programmers therefore have to make a conscious effort to write programs which can easily be run on different configurations.

3 3 3 NON DETERMINACY

Sequential programmers are used to the idea of a bug. There are solid bugs and intermittent bugs. Intermittent bugs are data dependent, a program run on the same inputs will work every time or it fails every time. Concurrent programming has a third type of bug: the bug that depends on the relative timing of concurrent processes. These are very often not repeatable, even if the program is rerun on the same data.

The problem arises because all the processors are allowed to run at their own speed. *There is no attempt to constrain the processors into a lock step.* Thus the order of events can change from one test run to another. It is obviously important at the design phase not to assume an exact ordering of the events. One would also expect it to be impossibly hard to test and debug Occam software, yet programmers commonly do succeed. The key to success is to reduce the need for testing to an absolute minimum, and to understand exactly what Occam defines as the effect of a program and what it leaves undefined. Occam is as precise as any other language about what individual processes do, but it cannot specify the relative timing of concurrent processes (otherwise the whole point of

concurrency would be lost)

It is the programmer's responsibility to ensure that when a program terminates it has completed the required function, regardless of the order in which things happened between starting and termination. Since Occam does not guarantee determinacy, it has been designed to express and handle non-determinacy very simply, flexibly, and elegantly.

3 3 4 DEADLOCK

A classic problem in concurrent systems is deadlock. This is an affliction whereby one part of a program is waiting for another to do something, the other is waiting for the first to do something else, and since both are waiting, neither can do what other expects. There may, of course, be a set of processes involved, rather than a pair.

All Transputer deadlocks are essentially the same in that they involve a closed chain of processes, each trying to communicate with another, but with no pair of them willing to participate in any one communication. There is an enormous variety of ways which may lead to deadlock, and that makes it hard to avoid. It is also difficult to analyse and hard to cure after it is found to occur in a program. Formal methods used are writing only simple code, supported by intuition and back-of-envelope sketches.

3 3 5 GENERAL ALGORITHM DESIGN CONSIDERATIONS

Most of the concurrent algorithms are only loosely related to their sequential equivalents. The conversion of a sequential algorithm into concurrent algorithm, often called parallelization, is too ill defined and difficult to be automated and is often attempted by hand. The following points should be given due consideration while developing the concurrent algorithms.

Granularity

The term granularity refers to the size of the task that are distributed as concurrent processes. In a matrix multiplication, all the elements of the result can, in principle, be

evaluated concurrently because none of the calculations depends on the result of any other. That would be fine grain concurrency. A more coarse grain division of labour could evaluate one quarter of the result, working sequentially on one processor, while evaluating the other three quarters on three other processors.

Intuitively, one would expect fine-grained concurrency to deliver results faster but to use more processors. With the Transputers, a finer-grain division of work requires more information to be passed between processors. Reducing the grain size below some optimum value for a particular problem can incur message passing costs which grossly outweigh the expected benefits.

Granularity is a particular problem in the conversion of existing sequential code into concurrent methods. It is relatively easy to recognize fine-grain potential parallelism, e.g. several assignments whose order is immaterial, or a simple loop. It is much harder to identify the larger units of a program which might safely be run concurrently.

Performance measure and efficiency

One measure of the quality of a concurrent system is its efficiency in using the processors. If a single processor solution takes n seconds, it is desirable to achieve a solution with m processors in n/m seconds. Complete efficiency is never attainable, but the user tries to get near to it. Efficiency must fall off as more processors are added.

Balancing communication and processing

Some times, the concurrent algorithm may show very poor efficiency. The reasons could be

The computation has not been divided equally — one of the processors still has to do far more than $1/n$ th of the work.

The processes are independent, they have to share or communicate information via links, and some of them spend a lot of time waiting for others, during which time they cannot continue with the computation.

Even if time is not wasted in waiting for messages, there may be a lot of time spent in

passing the messages

These three possibilities are quite distinct. The first one has to be solved by load balancing — attempting to equalize the computation load on each processor. The second can be described as '*spurious concurrency*'. It can occur by accident, and it is not easy to predict, analyse or detect. Even when it is known that it is happening, it is not easily cured. The third reason for inefficiency is a difficult optimization problem between computation and communication times.

Software development

Transputer software is mostly developed under the 'Transputer development environment' (TDS) supplied by INMOS. TDS and related systems provide an integrated environment for editing, compiling and running the programs. They are centered on the folding editor which embodies an elegant and general way of representing and handling large amount of Occam text within a small screen. They lack some of the support tools that are common in other systems (e.g., in UNIX, diff, grep, multitasking, batch files, aliases, email, login files).

The TDS is so much an integrated environment that its files are not easily handled in other systems, not even in systems such as MS-DOS which is acting as the host or file server for TDS. This makes it hard for the utility software on the host system to provide the facilities that are missing from the TDS.

Programmers would very much like to have further help in the peculiar difficulties of concurrent program development. The strongest demand at the moment, and the one which seems nearest to fulfillment, is for software tools for profiling, monitoring, and debugging.

Needs remaining unsatisfied

There are several areas in which user's experience has established a requirement that no hardware or software has yet fulfilled. These users need a more comprehensive (and

more familiar) software development environment, with tools for designing and constructing concurrent programs as well as more normal services for sequential programming, module management, etc. Extensive and usable monitoring and debugging facilities should be available, and more accessible formal methods, with software tools to support their use, would be welcome. The problem of file capacity, transfer rate and back-up have still to be solved.

On the hardware side, it is widely felt that the Transputer should use its on-chip memory as cache store, and that it should provide at least some support for memory management and protection. More links per Transputer would be welcome, as would automatic forwarding of messages between processors which are not directly connected.

Nevertheless the processing speed, the ease of multiprocessor interfacing and the availability of a naive programming language with attractive features make the Transputer a very good candidate for multiprocessing.

3.4 SETUP OF A TRANSPUTER NETWORK SYSTEM [7]

The major components of a processor network are

Host computer (mostly a PC)

Interface unit

processing element array (Transducers)

Interconnection networks

Host computer

The host computer is intended to provide system monitoring, data storage and management. It generates global control codes and object codes of processor elements. It can be a microcomputer, workstation, or a main frame. We have used PC-AT for this purpose. Processor elements can be accessed by a procedure call on the host, or through an interactive programmable command interpreter.

Interface unit

Interface unit is an interface between the host and PE array. Interface unit, connected to the host via host or host bus, or DMA has the function of downloading, uploading, buffering array data and handling interrupts. It supports high bandwidth communication (accompanying high-speed processing) between the array and the host. For balancing between low bandwidth of the system I/O and high bandwidth of the processor array, sufficient buffering is provided.

Processing element array

A PE array consists of a number of processing elements with local memory. PE's effectively utilize its data storage thereby saving communication time.

Interconnection network

Interconnection within PE array are provided by large switching networks which provide flexibility of interconnections and high speed communication between the PE's.

3.4.1 HARDWARE SETUP OF THE TRANSPUTER NETWORK SYSTEM [1,2]

A nine-node transputer network has been set up in the image processing lab at IIT Kanpur. The setup contains two IMS B003 evaluation boards, each having four transputers and an IMS B004 transputer system development (TDS) board having one transputer. Of these nine transputers, five are T800's and four are T414's. The host is an PC-AT (80386). One of the transputers functions as the interface unit, known as root processor, with 4 Mbytes of on board RAM. The root is connected to the host via a link adapter (IMS C012). Most of the time root transputer will be used for executing the I/O routines required for the host file server. PEs are individual transputers each with 256 Kbytes of RAM on board. Transputer links are connected by a programmable switch known as Link switch.

Link adapter

It connects the host and the root transputer. It provides for full duplex transputer link communication by converting bi-directional serial link data into parallel

data stream

Host

The host serves as a file server as memory management and file system are not supported by transputers. This facility of host is accessed by a program called **server**. The server reads a DOS file to determine the network configuration, the programs to be loaded and the boot order. The host loads the network via the root transputer by sending loading information to it, which in turn boots and loads the transputers connected to it which again passes the and loading information forward and so on until the whole network has been booted and loaded.

Booting of the transputer network

A communication protocol exists between the host and the transputer network to direct the code to the desired place in each transputer. The bootstrap code for each transputer is sent first. After all transputers are booted, the code of each of the procedures allocated to processors is exported to the network preceded by necessary routing and loading information. Following this, the code which calls the procedures is sent to each processor.

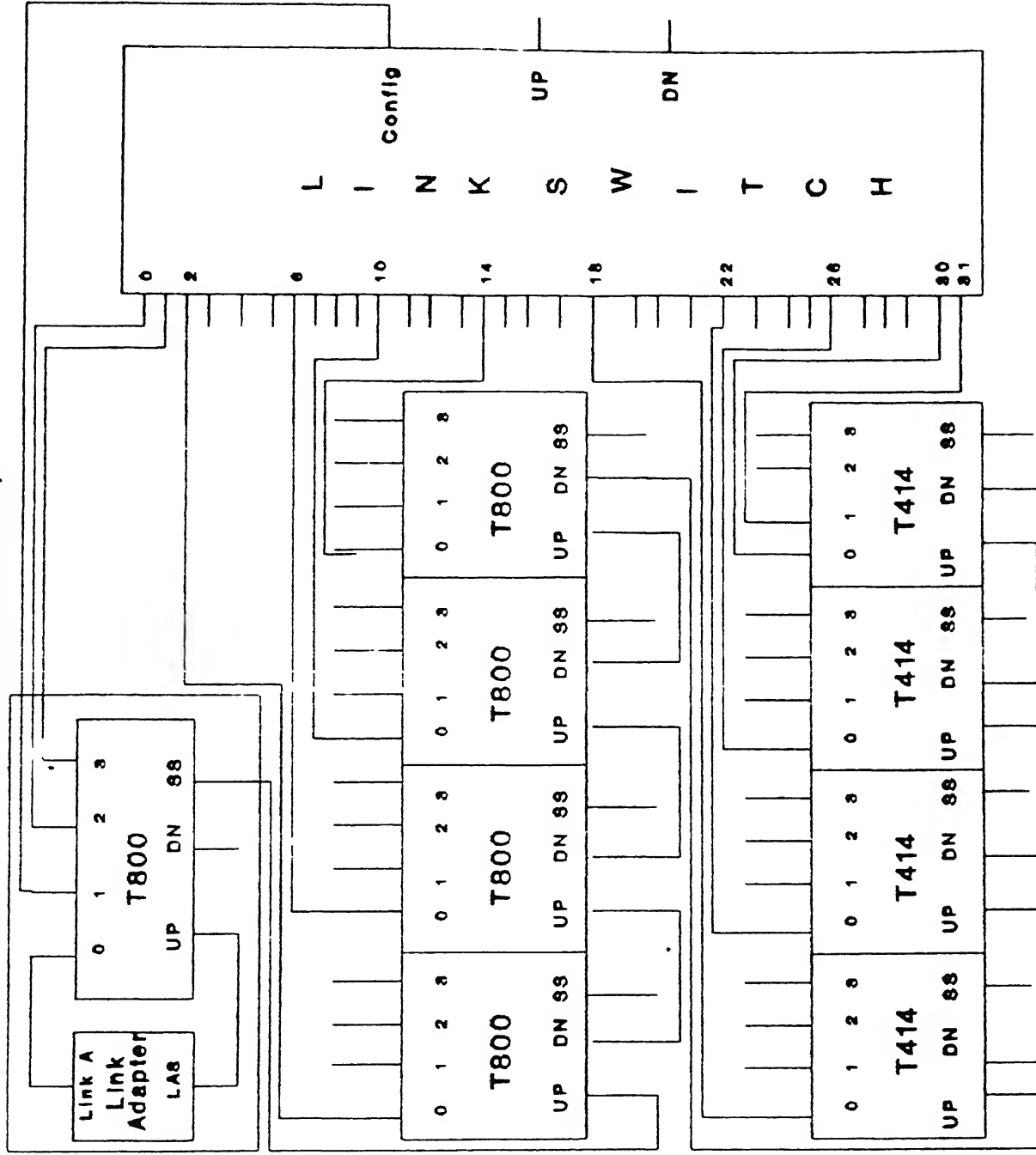
Link switch

Transputer network is interconnected using Link switch. It is a programmable link switch designed to provide a full crossbar switch between 32 link inputs and 32 link outputs. It uses the capabilities of VLSI to offer simple, easy to use and cheap interconnections for computer systems. It introduces on the average of 1.75 bit time delay on the signal. The switch is programmed via a separate link called the configuration link. In the setup, LINK0 of the root transputer is connected to the host and LINK1 is used as the configuration link for the link switch, so only LINK2 and LINK3 on the root transputer are free. Also, since the link switch supports only 32 connections, two links of the last (9th) transputer cannot be used at present.

HOST
(AT-386)

TRANSPUTER
Network

Hardware Setup



3 4 2 CODE DEVELOPMENT [1]

Occam is used for the program development under the TDS environment. The same program can be developed on a single node or on a multi node transputer network. On single node program would be developed as a parallel program of n processes. It would be using soft channels for communication between the parallel processes. On multi node network some changes have to be made

- 1 The individual processes are to be define as procedures whose parameters should be only the hard channels and should be compiled separately

- 2 The separately compiled procedures are 'PLACED' on individual transputers and the program is then compiled to generate a network code file

- 3 For running the program, network link switch should be configured

The single/multi node program can be tested from the TDS environment or it could be a standalone program bootable by the external host by using the alien file server routines available in the TDS environment

CHAPTER 4

ALGORITHMS FOR CALCULATING COEFFICIENTS

In this chapter, we are going to study and implement different algorithms for computing Galois Transform coefficients. All the algorithms have been implemented on Transputer facility available in IP-LAB. Also, time comparison of these algorithms has been done for same input function. The algorithms are

- (1) USING SYSTOLIC ARRAY
- (2) POLYNOMIAL COMPUTATION USING HORNER'S RULE
- (3) FFT (USING LINEAR ARRANGEMENT OF PROCESSORS)
- (4) FFT (USING MESH ARRANGEMENT OF PROCESSORS)

4.1 USING SYSTOLIC ARRAY [7,10]

According to H.T. Kung, "a systolic system is a network of processors which rhythmically compute and pass the data through the system". Once a data item is brought out from the memory, it can be used effectively at each cell it passes while being "pumped" from cell to cell along the array.

The computational tasks can be classified into two families

1. Compute-bound computation
2. I/O bound computation

In a computation, if the total number of operations is larger than total number of I/O operations, then the computation is compute-bound, otherwise it is I/O bound. Speeding up the I/O bound computation requires an increase in memory bandwidth, which is difficult in current technologies. Speeding up the compute-bound computation, however

may be accomplished by using Systolic arrays

The main features of systolic array are

Synchronicity

Modularity and regularity The array consists of modular processing elements with homogeneous interconnections. Also, the network may be extended indefinitely.

Spatial and temporal locality The array manifests a locally communicative interconnection structure, i.e. spatial locality. There is at least one unit delay so that signal transactions from one node to next node can be completed i.e. temporal locality.

Pipelability The array exhibits a linear rate pipelability i.e. it should achieve $O(M)$ speedup, in terms of the processing rate, where M is the number of processing elements.

The major factors for adopting Systolic arrays for special architectures are

- 1 Simple and regular design
- 2 Concurrency and communication
- 3 Balancing computation with I/O

4.1.1 DESIGN METHODOLOGY FOR SYSTOLIC ARRAYS

Due to fast progress in VLSI technology, algorithm oriented array architectures appear to be effective, feasible and economic. Therefore a systematic methodology of mapping computations onto systolic array has been developed.

Parallel algorithm expression may be derived by two approaches

- 1 Vectorization of sequential algorithm expressions
- 2 Direct parallel algorithm expressions, such as snapshots, recursive equations, parallel codes, single assignment code, dependence graphs and so on.

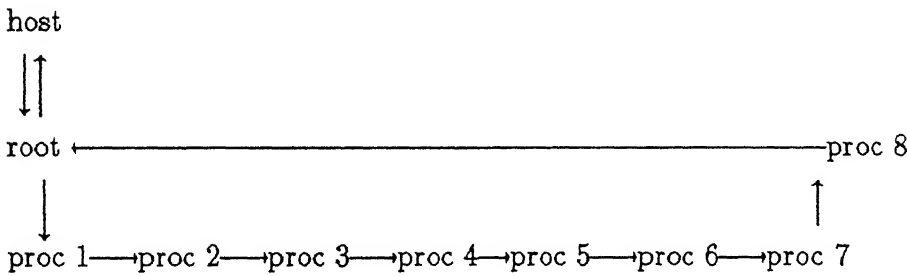
4.1.1.1 SYSTOLIC ARRAY DESIGN FOR MATRIX MULTIPLICATION [7]

As conceptually, we have to do matrix multiplication to find Galois coefficients we

will be designing systolic array for that If A and B are $N \times N$ matrices then their product is given by $C=AB$

The elements of C are $C_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{kj}$

This equation implies that all the multiplications involved can be carried out at the same time since they do not have any dependence between each other To get maximum parallelism, we need to propagate input data to N^3 multipliers, two input links for each multiplier That means at least $2N^3$ communication links are necessary As we only have eight processors and the size of the array can go up to 256×256 we will have to implement it in another way What we have done is to divide the Galois matrix $2 \times 2 \times 1$ into eight equal parts (row wise), and rows of input array are sent to each processor one by one In each processor vector multiplication takes place Processors are arranged in a linear manner as shown in the diagram below



Example

Let us take $k = 4$ so that $\zeta = 15$ The Galois matrix is

$$\begin{bmatrix} f(0) \\ f(1) \\ f(\alpha) \\ f(\alpha^2) \\ \\ \\ f(\alpha^{15}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 \\ 1 & 1 & 1 & & 1 & 1 \\ 1 & 1 & \alpha^{15} & & \alpha^2 & \alpha \\ 1 & 1 & \alpha^{14} & & \alpha^4 & \alpha^2 \\ & 1 & 1 & & & \\ & 1 & 1 & & & \\ 1 & 1 & \alpha & & \alpha^{14} & \alpha^{15} \end{bmatrix} \begin{bmatrix} a_{\alpha^{-\infty}} \\ a_{\alpha^0} \\ a_{\alpha} \\ a_{\alpha^2} \\ \\ \\ a_{\alpha^{15}} \end{bmatrix}$$

In each of the eight processors we place two rows from the above sixteen rows of Galois matrix. After this one by one each row of input array is sent into the network. In each processor two vector multiplication takes place and input row and output is transferred to the next processor.

4.1.2 ALGORITHM FOR SYSTOLIC ARRAY IMPLEMENTATION

Steps in the implementation of sequential algorithm are

- 1 Read the input data in a array $a[i][j]$

- 2

- a Start the computation for $i=0$ i.e send i th row (here first row) of input. Multiply the row with Galois matrix given in section 2.1

- b Receive the output in array b

- 3 Repeat step 2 for all values of $i=1, 2, \dots, 2^k-1$ and put the values in b

- 4 Transform the array $b[i][j]$

- 5 Repeat the steps 2,3 and 4 with b as input array and c as output array

In Computation of step 2, as each row of Galois matrix can be independently multiplied to input array, therefore all elements of a row of matrix b can be calculated independently. The algorithm for any processor in a transputer network can be given as

- 1 Set up $2^k/8$ rows of Galois matrix in the processor

- 2 Receive the row of input array and output elements from the previous processor. Multiply this row with part of Galois matrix present in the processor and calculate $2^k/8$ elements

- 3 Send the received input row and *total output* (output from previous processor plus output computed in this processor) to next processor. Go to step 2 again till the whole input array has passed through it

Once the whole array has passed through the network take the transpose and steps 1,2 and 3 are repeated once again

4.2 HORNER'S RULE IMPLEMENTATION [5]

The Horner's rule is a standard method for polynomial computation, irrespective of the type of polynomial under consideration. To describe the method of computation, let us consider the GP

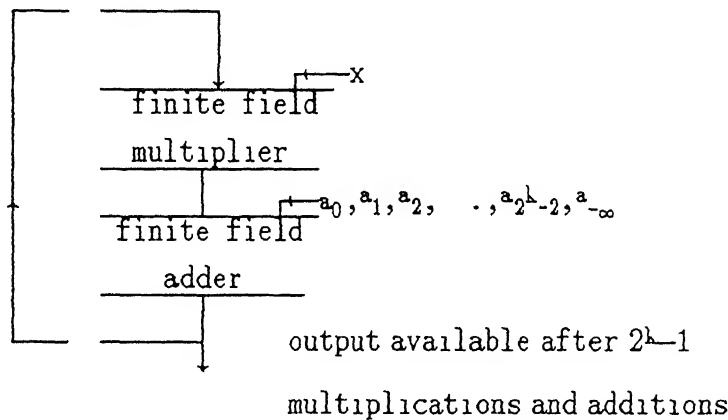
$$f(x) = a_{-\infty} + a_0 x^{2^k-1} + a_1 x^{2^k-2} + \dots + a_{2^k-1} x + a_{\zeta} x$$

where $\zeta = 2^k - 2$

This may also be written as

$$f(x) = (((((a_0 x + a_1)x + a_2)x + a_3)x + \dots + a_{\zeta-1})x + a_{\zeta})x + a_{-\infty} \quad (4.2.1)$$

The above equation suggests procedure for polynomial computation and is illustrated in figure given below



To get maximum parallelism, we need N^2 closed loop multiplier/adder circuits of above type. As we only have eight processors and the size of the array can go up to 256×256 we divide the work in 8 transputers only i.e. each Transputer will be doing the work of $N^2/8$ such circuits.

As we are dealing with two variable polynomials therefore we will have to apply the above rule twice. As we know

$$f(x_1, x_2) = \sum_{j_1, j_2} a_{j_1 j_2} x_1^{j_1} x_2^{j_2} \quad (4.2.2)$$

(1) Keeping j_1 constant (same row) we first calculate $\sum_{j_2=1}^{2^{k_2}-1} a_{j_1 j_2} x^{-j_2}$ for each value of x_2 (using Horner's rule). Let us define output obtained for some value of constant j_1 and for $x_2 = \alpha^{j_2}$ as $b_{j_1 j_2}$. Thus we obtain array b .

(2) Keeping j_2 constant (same column) we calculate $\sum_{j_1=1}^{2^{k_1}-1} b_{j_1 j_2} x^{-j_1}$ for each value of x_1 (using Horner's rule). Let us define output obtained for some value of constant j_2 and for $x_1 = \alpha^{j_1}$ as $c_{j_1 j_2}$, which is the required result.

4.2.1 ALGORITHM FOR POLYNOMIAL COMPUTATION

Steps in the implementation of sequential algorithm are

1 Read the input data in a array $a[i][j]$

2

a Start the computation for $i=0$ i.e. put the values of i th (here first) row as a 's in equation 4.2.1

b Now take $x = \alpha^{-m} = 0$ in equation 4.2.1 and compute the value and put in $b[0][0]$

c Repeat step 2 b for all values of $x = \alpha^{j-1}$ ($j=1, 2^{k_2}-1$) and put in $b[0][j]$

3 Repeat step 2 for all values of $i=1, 2^{k_1}-1$ and put the values in $b[i][j]$

4 Transform the array $b[i][j]$

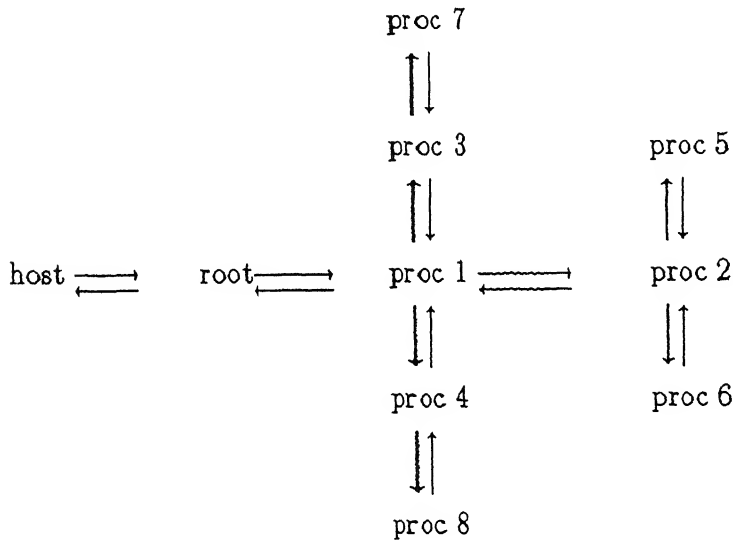
5 Repeat the steps 2,3 and 4 with b as input array and c as output array

Computation of step 3 does not depend on output of step 2, therefore all rows of array b can be calculated independently. Hence we will be calculating different rows concurrently on different processors. The algorithm for any processor in a transputer network can be given as

1 Receive the input row and compute Eq 4.2.1 for different values of x

2 Send the computed row back and receive next row

The root processor reads input values and sends them to the first processor. The first processor sends them to all processors, which compute and send back computed values to the first processor, which in turn sends them to the root processor. Here transpose is taken and the above steps are repeated once again. We can show the processor arrangement through a diagram given below.



4.3 FAST FOURIER TRANSFORM

As we know Galois coefficients are basically Fourier coefficients of function, therefore we can use the FFT algorithm. In the case of finite fields the size of DFT matrix is $2^k - 1 \times 2^k - 1$, therefore we can not use Butterfly algorithm (which can only be used for size 2^k). Hence we are going to use Cooley Tukey algorithm.

4.3.1 COOLEY TUKEY ALGORITHM [6]

This FFT algorithm is based on the strategy of changing a one dimensional Fourier transform into a two dimensional Fourier transform which is easier to compute. The

fourier transform of a vector v

$$V_k = \sum_{i=0}^{n-1} w^{ik} v_i \quad \text{here } w \text{ is the primitive element}$$

as it is written requires on the order of n^2 multiplications and n^2 additions. If n is not a prime number then this 1-D FFT can be converted to a 2-D FFT. This changes the computation to a form that is much more efficient. To understand the Cooley Tukey algorithm suppose that

$$n = n_1 n_2$$

$$i = i_1 + n_1 i_2$$

$$k = n_2 k_1 + k_2$$

$$\text{where } i_1 \text{ and } k_1 = 0, 1, \dots, n_1 - 1$$

$$\text{and } i_2 \text{ and } k_2 = 0, 1, \dots, n_2 - 1$$

Putting the above suppositions in the formula we get

$$V_{n_2 k_1 + k_2} = \sum_{i_2=0}^{n_2-1} \sum_{i_1=0}^{n_1-1} w^{(i_1 + n_1 i_2)(n_2 k_1 + k_2)} v_{i_1 + n_1 i_2}$$

Expand the product and because $w^n = 1$ therefore $w^{n_1 n_2 i_2 k_1}$ can be dropped. Also we define the convention that

$$v_{i_1 i_2} = v_{i_1 + n_1 i_2} \text{ and}$$

$$V_{k_1 k_2} = V_{n_2 k_1 + k_2}$$

In this way input and output data vectors are mapped into two dimensional arrays

In terms of two dimensional variables the formula becomes

$$V_{k_1 k_2} = \sum_{i_1=0}^{n_1-1} (w^{n_2})^{i_1 k_1} \left[w^{i_1 k_2} \sum_{i_2=0}^{n_2-1} (w^{n_1})^{i_2 k_2} v_{i_1 i_2} \right] \quad (4.3.1.1)$$

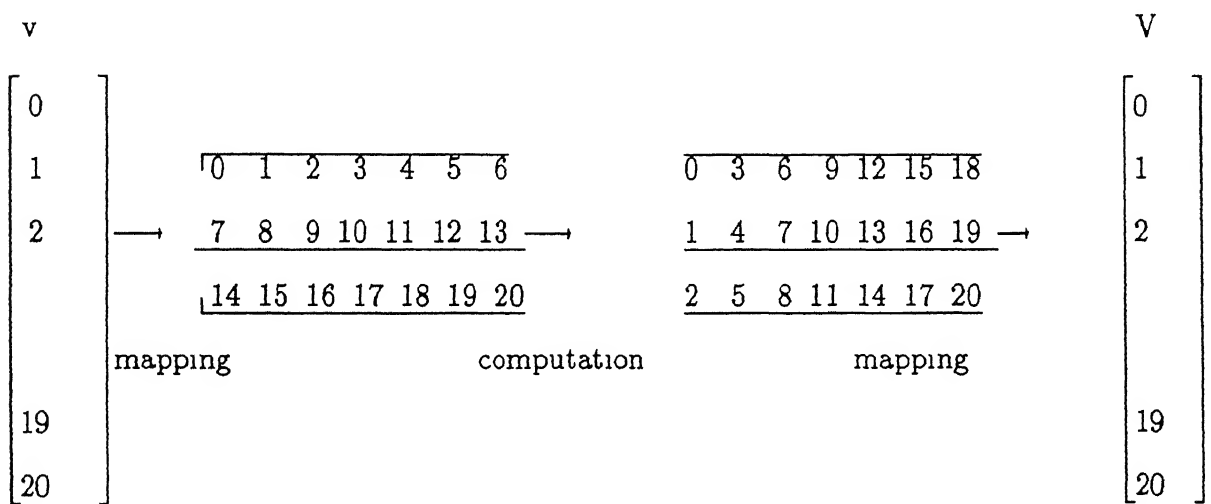
i.e. we have to first calculate $w^{i_1 k_2} \sum_{i_2=0}^{n_2-1} (w^{n_1})^{i_2 k_2} v_{i_1 i_2}$ for each i_1 and k_2 (therefore $(n_1 n_2^2 + n_1 n_2)$ multiplications required). Let the result be $u_{i_1 i_2}$. Later for each k_2 you have to find $\sum_{i_1=0}^{n_1-1} (w^{n_2})^{i_1 k_1} u_{i_1 i_2}$ ($n_1^2 n_2$ multiplications required).

Therefore number of multiplications $\cong n(n_1 + n_2) + n$

Number of additions $\cong n(n_1 + n_2 - 2)$

Example

The cooley tukey FFT can be visualized as mapping a 1-D array into a 2-D array as shown below for $n = 21$. The computation consists of 3 point DFT on each column, followed by element by element multiplication throughout the new array by w^{1k2} , followed by an 7 point DFT on each row. Observe that the components of the transform V are found arranged differently in the array than the components of the signal v this is known as address shuffling



4.3.2 Algorithm for FFT (in 2-D case)

Steps in the implementation of the sequential algorithm are

- 1 Read the input data in a array $a[i][j]$
- 2 Start the computation for $i=0$ i.e take the i th (here first) row Take factors of $n=n_1n_2$ and implement the equation 4.3.1.1 and put the results in $b[0][j]$ (first row)
- 3 Repeat step 2 for all values of $i=1,2, \dots, 2^{h_1}-1$ and put the values in $b[i][j]$
- 4 Transform the array $b[i][j]$
- 5 Repeat the steps 2,3 and 4 with b as input array and c as output array

Computation of step 3 does not depend on output of step 2, therefore all rows of array b can be calculated independently Hence we will be calculating different rows

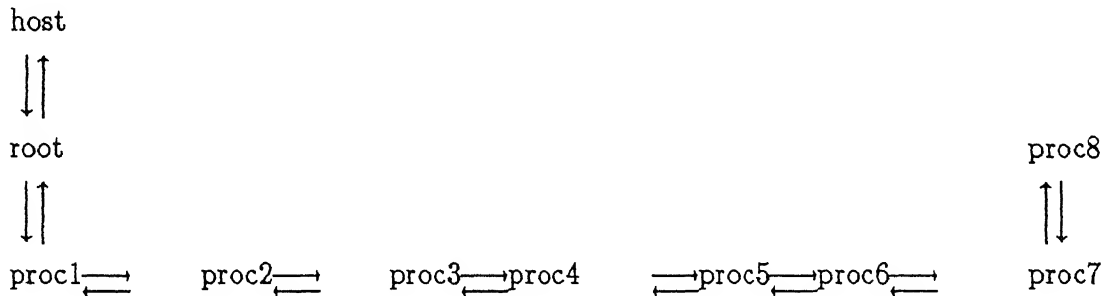
concurrently on different processors. The algorithm for any (kth) processor in a transputer network can be given as

1 Receive kth row and Compute Eq 4.3.1.1

2 Send the computed row back and receive next row(k+8). Repeat the step 2

We have implemented the above algorithm with two different arrangements of processors

In linear arrangement of processors each processor computes and sends back computed values to its preceding processor and thus computed values reach root processor via the same path in opposite direction. Here transpose is taken and the data is again sent in the same way. The arrangement can be shown in the diagram given below



In the mesh arrangement the root processor reads input values and sends them to the first processor. The first processor sends them to all processors, which compute and send back computed values to the first processor, which in turn sends them to the root processor. Here transpose is taken and the above steps are repeated once again. The arrangement of processors is same as shown for Horner's rule.

CHAPTER 5

IDENTIFICATION OF ENGLISH ALPHABET AND ARABIC NUMERALS

5.1 INTRODUCTION

In previous chapters, we have developed the Algorithms for finding 2-D Galois transform (GT) coefficients from images and vice-versa. We are going to study the use of these coefficients and real images in the process of identification. Up till now, the methods which have been developed for identification use the segmentation techniques like line and edge detection for image enhancement and then pattern recognition techniques for image recognition. The method that we are going to use is based on matching of Galois coefficients of image with Galois coefficients stored in the computer memory. Although this type of method would require a huge memory, we are not going to compare this method with existing ones or test its viability. We are only trying to devise some method which is theoretically possible. As some criterion is necessary to evaluate its performance, we are comparing it with method which is based on matching of images (termed as real space matching in remaining part of thesis).

As the whole set of images would be too large a set to deal with, we are restricting our attention on 8×8 binary pixel representation of English alphabet and Arabic numerals. The GT coefficients would be taking values from $GF(2^3)$. In all, there are sixty two different shapes (twenty six for small alphabet, twenty six for bigger alphabet and ten for arabic numerals). The images of these English alphabet and Arabic numerals (jointly termed as alphabet in remaining part of thesis) are given in Appendix A. These images are of the IBM font of alphabet used in PC. Some examples of images are

a	3	K
0 0 0 0 0 0 0 0	0 1 1 1 1 1 0 0	1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0	1 0 0 0 0 0 1 0	1 0 0 0 1 0 0 0
1 1 1 1 1 1 0 0	0 0 0 0 0 0 1 0	1 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0	0 1 1 1 1 1 0 0	1 1 1 0 0 0 0 0
0 1 1 1 1 1 1 0	0 0 0 0 0 0 1 0	1 0 0 1 0 0 0 0
1 0 0 0 0 0 1 0	1 0 0 0 0 0 1 0	0 0 0 0 1 0 0 0
1 1 1 1 1 1 1 0	0 1 1 1 1 1 0 0	1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

The coefficients of different alphabet are given in Appendix B Corresponding coefficients of above alphabet are

a	3	K
0 1 3 5 7 2 4 6	0 1 4 7 4 6 6 7	1 0 5 2 6 3 7 4
0 0 0 0 0 0 0 0	0 1 4 7 4 6 6 7	1 1 6 4 3 7 2 5
4 3 5 0 1 7 2 4	7 0 2 0 6 2 6 1	7 3 5 3 4 5 2 4
7 5 6 2 3 0 7 1	6 0 3 3 4 0 1 4	6 5 2 2 3 5 7 7
6 7 2 1 3 2 6 0	2 4 7 0 4 7 4 6	5 7 5 7 5 2 1 2
6 2 0 4 6 3 1 5	4 0 0 5 1 5 7 7	4 2 2 3 6 3 6 5
7 4 1 5 0 5 2 7	5 6 0 4 7 4 6 6	3 4 4 5 5 3 3 1
4 6 3 3 4 1 0 5	3 7 6 6 7 0 4 7	2 6 3 2 1 6 3 2

Here 0 stands for 0, 1 stands for 1, 2 stands for α , 3 stands for α^2 , 4 stands for α^3 and 7 stands for α^6 where α is the primitive element of $GF(2^3)$

As the size of possible shapes is limited to sixty two and in we are only taking binary representation of images therefore we should require atleast six comparisons (because $2^6=64 > 62$) to identify in real space (a tree like structure) Similarly for coefficient space as the the number of levels is eight therefore we should require atleast two comparisons (because $8^2=64 > 62$) for identification Thus we see that simple mathematics tells us that we are at advantage (as regards to number of comparisons or time required for identification) in coefficient domain But we are at disadvantage in coefficient domain because of the time required for calculation of coefficients (assuming data is being transmitted and stored in the form of real space) Let's see whether this trade off in time will be beneficial (i.e less time is required) for identification in real sample domain or in coefficient domain We have developed three methods for identification These methods

have been developed somewhat in an ad-hoc manner only for the purpose of comparing the performances of identification in sample domain and in coefficient domain. The main motivation behind methods developed has been to try to minimize the average time required for identification of any alphabet. We have tried to keep in mind the probability of occurrence of different alphabet in devising the methods. The method should be such that in the calculation of average time required for identification, alphabet like t,e,a should have less comparisons required for identification than x,q,z. But I do not contend that the methods developed are of least average time in identification of above sixty two shapes. For developing those methods one would require a deep study of operations research. Also our basic aim in this thesis is different than that. In later part of this chapter we will fleetingly gloss over the problem that what should be the ideal structure of alphabet (i.e. their shapes) to have quick identification in both domain.

Of the three methods

Two methods are based on the differences in coefficient domain and

One method is based on the differences in real domain

We are going to study the Algebra behind the three methods. Also we will study the three (methods) algorithms in detail. These algorithms have been developed on transputer facility available in I P lab.

5.2 ALGEBRAIC PROPERTIES OF GALOIS COEFFICIENTS in 8*8 MATRICES

As we study the different coefficients, we observe some Properties of Galois coefficients. We have observed these coefficients satisfy the conjugacy constraints as given in example 2.2.1. Conjugacy classes have been enumerated here again. In each class each element is square of preceding element.

$$(1) \{(-\infty, -\infty)\} \quad (2) \{(-\infty, 0)\}$$

$$(3) \{(0, -\infty)\} \quad (4) \{(0, 0)\}$$

$$(5) \{(-\infty, 1), (-\infty, 2), (-\infty, 4)\}$$

$$(6) \{(-\infty, 3), (-\infty, 6), (-\infty, 5)\}$$

$$(7) \{(0, 1), (0, 2), (0, 4)\}$$

$$(8) \{(0, 3), (0, 6), (0, 5)\}$$

- | | |
|---|-----------------------------------|
| (9) $\{(1, -\infty), (2, -\infty), (4, -\infty)\}$ | (10) $\{(1, 0), (2, 0), (4, 0)\}$ |
| (11) $\{(1, 1), (2, 2), (4, 4)\}$ | (12) $\{(1, 2), (2, 4), (4, 1)\}$ |
| (13) $\{(1, 3), (2, 6), (4, 5)\}$ | (14) $\{(1, 4), (2, 1), (4, 2)\}$ |
| (15) $\{(1, 5), (2, 3), (4, 6)\}$ | (16) $\{(1, 6), (2, 5), (4, 3)\}$ |
| (17) $\{(3, -\infty), (6, -\infty), (5, -\infty)\}$ | (18) $\{(3, 0), (6, 0), (5, 0)\}$ |
| (19) $\{(3, 1), (6, 2), (5, 4)\}$ | (20) $\{(3, 2), (6, 4), (5, 1)\}$ |
| (21) $\{(3, 3), (6, 6), (5, 5)\}$ | (22) $\{(3, 4), (6, 1), (5, 2)\}$ |
| (23) $\{(3, 5), (6, 3), (5, 6)\}$ | (24) $\{(3, 6), (6, 5), (5, 3)\}$ |

The four coefficients given by

$$(1) \{(-\infty, -\infty)\} \quad (2) \{(-\infty, 0)\} \quad (3) \{(0, -\infty)\} \quad (4) \{(0, 0)\}$$

are either 0 or 1

We can also find relationships between weighted sum of coefficients

For example take the sum of first row

$$a_{-\infty-\infty} + a_{-\infty 0} + a_{-\infty 1} + a_{-\infty 2} + a_{-\infty 3} + a_{-\infty 4} + a_{-\infty 5} + a_{-\infty 6}$$

$$\text{Because } a_{-\infty j} = \sum_x x^j f(0, x)$$

$$\text{Therefore } LHS = \sum_j \sum_x x^j f(0, x) + f(0, 0)$$

first term is for $j=0, 1, 2, 3, 4, 5, 6$ and second term is for $j=-\infty$

$$LHS = \sum_x f(0, x) \sum_j x^j + f(0, 0)$$

Because $\sum_j x^j$ is zero for all x except for $x=0$ and 1

ie for $x=\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$ $\sum_j x^j$ is 0 and for $x=0, 1$ it is 1

$$\text{Therefore } LHS = (f(0, 1) + f(0, 0)) + f(0, 0) = f(0, 1)$$

Take the sum of second row

$$a_{0-\infty} + a_{00} + a_{01} + a_{02} + a_{03} + a_{04} + a_{05} + a_{06}$$

$$\text{Because } a_{0j} = \sum_{x_1} \sum_{x_2} x_2^j f(x_1, x_2)$$

- | | |
|---|-----------------------------------|
| (9) $\{(1, -\infty), (2, -\infty), (4, -\infty)\}$ | (10) $\{(1, 0), (2, 0), (4, 0)\}$ |
| (11) $\{(1, 1), (2, 2), (4, 4)\}$ | (12) $\{(1, 2), (2, 4), (4, 1)\}$ |
| (13) $\{(1, 3), (2, 6), (4, 5)\}$ | (14) $\{(1, 4), (2, 1), (4, 2)\}$ |
| (15) $\{(1, 5), (2, 3), (4, 6)\}$ | (16) $\{(1, 6), (2, 5), (4, 3)\}$ |
| (17) $\{(3, -\infty), (6, -\infty), (5, -\infty)\}$ | (18) $\{(3, 0), (6, 0), (5, 0)\}$ |
| (19) $\{(3, 1), (6, 2), (5, 4)\}$ | (20) $\{(3, 2), (6, 4), (5, 1)\}$ |
| (21) $\{(3, 3), (6, 6), (5, 5)\}$ | (22) $\{(3, 4), (6, 1), (5, 2)\}$ |
| (23) $\{(3, 5), (6, 3), (5, 6)\}$ | (24) $\{(3, 6), (6, 5), (5, 3)\}$ |

The four coefficients given by

$$(1) \{(-\infty, -\infty)\} \quad (2) \{(-\infty, 0)\} \quad (3) \{(0, -\infty)\} \quad (4) \{(0, 0)\}$$

are either 0 or 1

We can also find relationships between weighted sum of coefficients

For example take the sum of first row

$$a_{-\infty-\infty} + a_{-\infty 0} + a_{-\infty 1} + a_{-\infty 2} + a_{-\infty 3} + a_{-\infty 4} + a_{-\infty 5} + a_{-\infty 6}$$

$$\text{Because } a_{-\infty j} = \sum_x x^j f(0, x)$$

$$\text{Therefore } LHS = \sum_j \sum_x x^j f(0, x) + f(0, 0)$$

first term is for $j=0,1,2,3,4,5,6$ and second term is for $j=-\infty$

$$LHS = \sum_x f(0, x) \sum_j x^j + f(0, 0)$$

Because $\sum_j x^j$ is zero for all x except for $x=0$ and 1

ie for $x=\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6$ $\sum_j x^j$ is 0 and for $x=0, 1$ it is 1

$$\text{Therefore } LHS = (f(0, 1) + f(0, 0)) + f(0, 0) = f(0, 1)$$

Take the sum of second row

$$a_{0-\infty} + a_{00} + a_{01} + a_{02} + a_{03} + a_{04} + a_{05} + a_{06}$$

$$\text{Because } a_{0j} = \sum_{x_1, x_2} x_1^j x_2^j f(x_1, x_2)$$

$$\text{Therefore L H S} = \sum_{x_1} \sum_{x_2} f(x_1, x_2) \sum_j x_2^j + \sum_{x_1} f(x_1, 0)$$

Because $\sum_j x_2^j$ is zero for all x_2 except $x_2 = 0$ and 1 therefore

$$\text{L H S} = \sum_{x_1} f(x_1, 1) \text{ i.e. sum of second column in function domain}$$

Let's take the weighted sum of third row What should be the weights? How to decide? Instead of $\sum_j x_2^j$ we should have a polynomial such that it is zero for all x_2 except $x_2 = \alpha$ Such a polynomial is

$$g(x) = (x-1)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)$$

$$g(x) = x^6 + \alpha x^5 + \alpha^2 x^4 + \alpha^3 x^3 + \alpha^4 x^2 + \alpha^5 x + \alpha^6$$

$$g(x) = 0 \text{ for } x = 1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6 \text{ and } g(x) = \alpha^6 \text{ for } 0 \text{ and } \alpha$$

Using the above property we observe that if we multiply the coefficients by above weightages then weighted sum of third row is

$$\alpha^6 a_{1-\infty} + \alpha^6 a_{10} + \alpha^5 a_{11} + \alpha^4 a_{12} + \alpha^3 a_{13} + \alpha^2 a_{14} + \alpha a_{15} + a_{16}$$

$$\text{Because } a_{1j_2} = \sum_{x_1} \sum_{x_2} x_1 x_2^{j_2} f(x_1, x_2)$$

$$\text{Therefore L H S} = \sum_{x_1} \sum_{x_2} x_1 f(x_1, x_2) g(x_2) + \alpha^6 \sum_{x_1} x_1 f(x_1, 0)$$

first term is for $j_2 = 0, 1, 2, 3, 4, 5, 6$ and second term is for $j_2 = -\infty$

first term is zero for all x_2 except for $x_2 = 0$ and α

$$\text{Therefore L H S} = \sum_{x_1} x_1 f(x_1, 0) \alpha^6 + \sum_{x_1} x_1 f(x_1, \alpha) \alpha^6 + \alpha^6 \sum_{x_1} x_1 f(x_1, 0)$$

$$\text{Therefore L H S} = \sum_{x_1} x_1 f(x_1, \alpha) \alpha^6$$

$$\alpha^6 a_{1-\infty} + \alpha^6 a_{10} + \alpha^5 a_{11} + \alpha^4 a_{12} + \alpha^3 a_{13} + \alpha^2 a_{14} + \alpha a_{15} + a_{16} = \alpha^6 \sum_{x_1} x_1 f(x_1, \alpha)$$

Multiplying both sides by α we get

$$a_{1-\infty} + a_{10} + \alpha a_{11} + \alpha^2 a_{12} + \alpha^3 a_{13} + \alpha^4 a_{14} + \alpha^5 a_{15} + \alpha^6 a_{16} = \sum_{x_1} x_1 f(x_1, \alpha)$$

Therefore weighted sum of third row is equivalent to third galois coefficient of vector $f(x_1, \alpha)$

Similarly we would get

$$\begin{aligned}
a_{2-\infty} + a_{20} + \alpha^5 a_{21} + \alpha^3 a_{22} + \alpha^6 a_{23} + \alpha^4 a_{24} + \alpha^2 a_{25} + \alpha^1 a_{26} &= \sum x_1^2 f(x_1, \alpha^2) \\
a_{3-\infty} + a_{30} + \alpha^4 a_{31} + \alpha^5 a_{32} + \alpha^2 a_{33} + \alpha^6 a_{34} + \alpha^3 a_{35} + \alpha^1 a_{36} &= \sum x_1^3 f(x_1, \alpha^3) \\
a_{4-\infty} + a_{40} + \alpha^3 a_{41} + \alpha^6 a_{42} + \alpha^2 a_{43} + \alpha^5 a_{44} + \alpha^4 a_{45} + \alpha^1 a_{46} &= \sum x_1^4 f(x_1, \alpha^4) \\
a_{5-\infty} + a_{50} + \alpha^2 a_{51} + \alpha^4 a_{52} + \alpha^6 a_{53} + \alpha^3 a_{54} + \alpha^5 a_{55} + \alpha^1 a_{56} &= \sum x_1^5 f(x_1, \alpha^5) \\
a_{6-\infty} + a_{60} + \alpha^1 a_{61} + \alpha^2 a_{62} + \alpha^3 a_{63} + \alpha^4 a_{64} + \alpha^5 a_{65} + \alpha^6 a_{66} &= \sum x_1^6 f(x_1, \alpha^6)
\end{aligned}$$

Similar relations can be obtained for weighted sum of columns

$$\begin{aligned}
a_{-\infty\infty} + a_{0\infty} + a_{1\infty} + a_{2\infty} + a_{3\infty} + a_{4\infty} + a_{5\infty} + a_{6\infty} &= f(1, 0) \\
a_{-\infty 0} + a_{00} + a_{10} + a_{20} + a_{30} + a_{40} + a_{50} + a_{60} &= \sum f(1, x_2) \\
a_{-\infty 1} + a_{01} + \alpha^6 a_{11} + \alpha^5 a_{21} + \alpha^4 a_{31} + \alpha^3 a_{41} + \alpha^2 a_{51} + \alpha^1 a_{61} &= \sum x_2 f(\alpha, x_2) \\
a_{-\infty 2} + a_{02} + \alpha^5 a_{12} + \alpha^3 a_{22} + \alpha^6 a_{32} + \alpha^4 a_{42} + \alpha^2 a_{52} + \alpha^1 a_{62} &= \sum x_2^2 f(\alpha^2, x_2) \\
a_{-\infty 3} + a_{03} + \alpha^4 a_{13} + \alpha^6 a_{23} + \alpha^2 a_{33} + \alpha^5 a_{43} + \alpha^3 a_{53} + \alpha^1 a_{63} &= \sum x_2^3 f(\alpha^3, x_2) \\
a_{-\infty 4} + a_{04} + \alpha^3 a_{14} + \alpha^6 a_{24} + \alpha^2 a_{34} + \alpha^5 a_{44} + \alpha^4 a_{54} + \alpha^1 a_{64} &= \sum x_2^4 f(\alpha^4, x_2) \\
a_{-\infty 5} + a_{05} + \alpha^2 a_{15} + \alpha^4 a_{25} + \alpha^6 a_{35} + \alpha^3 a_{45} + \alpha^5 a_{55} + \alpha^1 a_{65} &= \sum x_2^5 f(\alpha^5, x_2) \\
a_{-\infty 6} + a_{06} + \alpha^1 a_{16} + \alpha^2 a_{26} + \alpha^3 a_{36} + \alpha^4 a_{46} + \alpha^5 a_{56} + \alpha^6 a_{66} &= \sum x_2^6 f(\alpha^6, x_2)
\end{aligned}$$

These are the relations that we are going to use to distinguish between different alphabet

5.3 METHOD 1 IDENTIFICATION IN COEFFICIENT SPACE

On observing the coefficient space for alphabet we find that in this space diagonal entries are mostly different for different alphabet. Therefore our first method would be based on diagonal entries. In this method initially we are going to compare elements a_{11} and a_{33} . Other elements i.e. a_{22}, a_{44}, a_{55} and a_{66} are conjugate elements of these two

elements On the basis of this comparison we get the following table In the table in those entries where more than one alphabet is present we compare a $_{1-\infty}$ Table obtained after these three comparisons is

Here 0 stands for 0, 1 stands for 1, 2 stands for α , 3 stands for α^2 , 4 stands for α^3 and 7 stands for α^6 where α is the primitive element of $GF(2^3)$

a_{11}	a_{33}	Alphabet	Alphabet(a_{10})	
0	0	D,U,Z	D(7),U(4),Z(6)	
0	1	v		
0	3	p		
0	4	w		
0	6	h		
0	7	c		
1	1	r		
1	3	a,x	a(7),x(0)	
1	4	g,N	g(5),N(7)	
1	6	l		
1	7	Q		
2	0	E		
2	1	1,4	1(0),4(4)	
2	4	l,T,8,W,B,3	l(0),T(0),8(1),W(4),B(7),3(7)	unidentified
2	5	q,J	q(3),J(2)	
2	6	o		
3	1	G,2	G(4),2(1)	
3	2	S		
3	5	H		
4	0	f		
4	2	e		
4	3	u		
4	4	j,n,7	j(6),n(2),7(1)	
4	5	M		
4	6	C,F	C(4),F(7)	
4	7	t		
5	0	b,5,6	b(7),5(0) 6(4),	
5	3	s,A	s(3),A(4)	
5	5	k,K,R	R(7),k(7),K(7)	unidentified
5	6	L,0	L(7),0(4)	
6	2	X		
6	3	d,9,O	d(2),O(6),9(3)	
6	6	m,V	m(4),V(5)	
7	0	Y		
7	1	P		
7	5	I		
7	6	y,z	y(7),z(0)	

After three comparisons only three cases are remaining where identification has not been completed. In these three cases a $-\infty$ is compared to finally identify all the remaining alphabet. Thus we observe that two to four comparisons of different coefficients is necessary to distinguish the alphabet in coefficient domain.

alphabet requiring two comparisons = 21

alphabet requiring three comparisons = 34

alphabet requiring four comparison = 7

Assuming equal probability of occurrence we would get

Average number of comparison = $(2 \times 21 + 3 \times 34 + 4 \times 7) / 62$

Therefore Average number of comparisons = 2.77 comparison

As we have tried to have lesser comparisons for more probable alphabet the real value would be lesser than 2.77.

5.4 METHOD 2 IDENTIFICATION IN COEFFICIENT SPACE

The above method has one defect. It is highly error prone. For example, if there is a change in coming alphabet at one position i.e. transmitted 0 is received as 1, then the coefficient space of this received alphabet can be entirely different from that of the transmitted alphabet. In the above method, each coefficient that we are going to match depends on the whole real space. If we choose the coefficient to be matched such that it depends only on one row or one column of real space, then chances of wrong identification are reduced. If we take the weighted sum of fourth row and fourth column in coefficient space, then we can get numbers that are only dependent on fourth column and fourth row of real space respectively. We have chosen the above rows for weighted addition because most of the information in sample domain is centered around them. The relations were explained in section 5.2. The pertinent relations have been written here again.

$$c1 = a_{-\infty} + a_{20} + \alpha^5 a_{21} + \alpha^3 a_{22} + \alpha a_{23} + \alpha^6 a_{24} + \alpha^4 a_{25} + \alpha^2 a_{26} = \sum_{x_1}^2 x_1^2 f(x_1, \alpha^2)$$

$$c2 = a_{-\infty 2} + a_{02} + \alpha^5 a_{12} + \alpha^3 a_{22} + \alpha a_{32} + \alpha^6 a_{42} + \alpha^4 a_{52} + \alpha^2 a_{62} = \sum_{x_2}^2 x_2^2 f(\alpha^2, x_2)$$

Here c1 and c2 are the modified coefficients to be compared The table obtained after these two comparisons is

Here 0 stands for 0, 1 stands for 1, 2 stands for α , 3 stands for α^2 , 4 stands for α^3 and 7 stands for α^6 where α is the primitive element of $GF(2^3)$

c1	c2	Alphabet
1	4	c,o,f,g
2	2	0
2	3	E
2	4	p,q,K,k
2	5	Z,i
2	6	d,9
2	7	3,5,6,8,b,B,S,G
3	4	r
4	0	C,L
4	1	J
4	2	u
4	4	v,y,D,O,Q,U,V
5	0	s
5	2	M,N
5	3	F
5	4	a,e
5	5	X,4,7
5	6	A
5	7	P,h,z,R
6	2	w
6	4	j,6
6	5	I,l,l,T
6	6	n
6	7	2
7	2	W,m,x
7	5	Y
7	7	t

After these two comparisons we compare weighted additions of different rows/columns (depending on value of c1 and c2) to identify still unidentified cases

Alphabet requiring two comparisons = 13

Alphabet requiring three comparisons = 35

Alphabet requiring four comparisons = 14

Assuming equal probability of occurrence we would get

Average number of comparisons = $(2 \times 13 + 3 \times 35 + 4 \times 14) / 62$

Average number of comparisons = 3 01 comparisons

Thus we observe that in this method the average number of comparisons has increased because we are comparing only a restricted portion of real space. Also the complexity of this method is more than the previous method.

5.5 METHOD 3 IDENTIFICATION IN REAL SPACE

In real space identification of alphabet, there are two methods. First one is based on six to eight successive comparisons of different points and second on comparison of additions of rows and columns. Second method has the advantage of error reduction because of the cancellation of two errors of opposite signs if they are occurring on the same row/column.

Second method Initially we count the number of 1's in the fourth row and fourth column as s_1 and s_2 . These two numbers are compared with their values stored in computer memory. After these two comparisons in those entries where complete identification has not been done we match additions of other rows/columns (depending on value of s_1 and s_2).

$$s_1 = f(3,0) + f(3,1) + f(3,2) + f(3,3) + f(3,4) + f(3,5) + f(3,6) + f(3,7)$$

$$s_2 = f(0,3) + f(1,3) + f(2,3) + f(3,3) + f(4,3) + f(5,3) + f(6,3) + f(7,3)$$

We get the following table

s_1	s_2	alphabet
1	1	j, X, L
1	2	C, J, 7
1	3	s, a, 2, z, Z
1	4	Y
1	6	i
1	7	l, 1, T, I
2	1	r, u, v, x, n, U, V
2	2	c, y, o, p, q, D, O, Q
2	3	e, g
2	7	4
3	1	M, N
3	2	K, k
3	3	w, 0
3	4	m

3	5	W
5	2	F
5	3	3,8,S,E,G
5	6	t
5	7	f
6	1	h
6	2	b,d,6,P,R
6	3	5,B,9
7	1	H
7	2	A

Alphabet requiring two comparisons = 11

Alphabet requiring three comparisons = 37

Alphabet requiring four comparisons = 12

Alphabet requiring five comparisons = 2

Assuming equal probability of occurrence we would get

Average number of comparisons = $(2 \times 11 + 3 \times 37 + 4 \times 12 + 5 \times 2) / 62$

Average number of comparisons = 3.08 comparisons

5.6 PERFORMANCE COMPARISON

Thus we observe that although number of comparisons required in this method is less than the number of comparisons required in successive comparison (Successive comparison of positions would have required at least 6 comparisons) method, it is more than that of the first method. On comparing the performance of three methods we find that least number of comparisons are required in first case but time wise third method is best as it does not require to calculate coefficients. What would be the effect as the size increases?

Let us assume that we have to identify all the possible shapes from a field of matrices of size $n \times n$ taking values from $GF(2)$. Then total number of input shapes possible is 2^{n^2} . The coefficient space would take values from $GF(n)$. Let us assume that we are using 8 node transputer facility to calculate GT coefficients. We also assume that we are using systolic array arrangement of processors. As we know that number of finite algebra multiplications/additions required to calculate Galois coefficients is $2 \times n^3$ where the size of

array is $n \times n$. As there are eight processors therefore, if time required for one finite algebra addition is k then

$$\text{total time required for calculation of coefficients is } = (2 \times n^3 \times k) / 8$$

As number of possible shapes is 2^{n^2} therefore in real space number of successive comparisons required to identify any shape is n^2 . Number of comparisons required in coefficient space is (n^2/m) where $m = \log_2 n$. If time required for one comparison is p then

$$\text{Total time required in identification in real space} = n^2 \times p$$

Total time required in identification in coefficient space $= (n^2 \times p) / m + (2 \times n^3 \times k) / 8$
 As second term (in second case) is increasing at a rate of n^3 therefore time required for calculation dominates over time required for comparison. Thus for identification purposes real space is better. But if we assume that we are also storing and transmitting data in coefficient form then identification in coefficient space would be better than identification in real space case.

What should be the ideal shapes of alphabet for quick identification?

As we have seen that for identification of sixty two alphabet we should have required only 2 comparisons (in coefficient space), but in our method we require 277 comparisons. This increase is due to the particular image/shape of alphabet. If there had been a set of two coefficients which is different for every alphabet we would have identified each alphabet in only two comparisons. As there is no such set, so to obtain such a set we would have to change coefficients (of that set) for some alphabet. We can choose any two coefficients in a set but we should also try to minimize changes. Therefore we will choose that set which has maximum number of two comparison identification. As change in any coefficient changes the whole image therefore we would get a new group of images. This group of shapes is not unique. Also this group may not work for identification in real space i.e. we may not get a set of six positions whose comparison will give every alphabet.

CHAPTER 6

SUMMARY AND CONCLUSIONS

6.1 RESULTS

We are going to compare the timings of different algorithms that were implemented on transputer facility to calculate the Galois Transform coefficients. Timings have been measured for different array sizes in sequential, parallel one node and parallel eight node configurations.

Timings for different array size (systolic array) are given below

ARRAY SIZE	SEQ	execution time USING SYSTOLIC ARRAY		Speedup
		PAR(1 node)	PAR(8 node)	
<u>n×n</u>				
4×4	5 2	5 7	3 9	1 33
8×8	49 3	53 1	17 1	2 88
16×16	498	547	107	4 65
32×32	5003	5500	845	5 92
64×64	49 6s	53 5s	7 1s	6 93
128×128	501s	540s	69s	7 26
256×256	75 4	80m	10 1m	7 46

s denotes time in seconds and m denotes time in minutes
All other timings are in milliseconds

We note that timings under each heading are increasing by about 10 times when we increase the size of array by 4 (i.e. when we double the n). Bulk of the time required for calculating coefficients goes in finite algebra additions. As the number of finite algebra additions is directly proportional to n^3 therefore when we increase the size by 2, then their number increases by eight. There is further increase in time due to increase in the number of finite algebra multiplications, thus jacking up the time by about 10.

CENTRAL LIBRARY
I.I.T., KANPUR
Acc. No. A. 115435

Timings for different array size FFT (linear arrangement) are given below

ARRAY SIZE	execution time			<u>Speedup</u>
	<u>SEQ</u>	<u>PAR(1 node)</u>	<u>PAR(8 node)</u>	
4×4	4 6	6 9	2 8	1 64
8×8	46 0	55 4	14 1	3 26
16×16	327 6	382 4	69 5	4 71
32×32	4561	5130	645	7 07
64×64	20 5s	23 9s	3 5s	5 86
128×128	395s	434s	55 6s	7 10
256×256	37 4m	38 3m	5 2m	7 19

s denotes time in seconds and m denotes time in minutes
All other timings are in milli seconds

Timings for different array size FFT (mesh arrangement) are given below

ARRAY SIZE	execution time			<u>Speedup</u>
	<u>SEQ</u>	<u>PAR(1 node)</u>	<u>PAR(8 node)</u>	
4×4	4 6	6 58	2 7	1 70
8×8	46	52 8	12 2	3 77
16×16	327 6	366	67 2	4 87
32×32	4561	4926	632	7 21
64×64	20 5s	22 6s	3 3s	6 21
128×128	395s	423s	54s	7 31
256×256	28 5m	30m	3 8m	7 51

s denotes time in seconds and m denotes time in minutes
All other timings are in milli seconds

On comparing the timings of linear and mesh arrangement of processors we observe that later one has better execution time and speedup factor because of less communication overhead time. In first and second case the average distance of each processor from root processor is

$$((1+2+3+4+5+6+7+8)/8) = 4 \text{ processors and}$$

$$((1+2+2+2+3+3+3+3)/8) = 2.37 \text{ processors respectively}$$

Timings for different array size (Horner's rule) are given below

ARRAY SIZE	SEQ	execution time USING HORNER'S RULE		Speedup
		PAR(1 node)	PAR(8 node)	
4×4	3.9	5.3	2.8	1.39
8×8	74.5	79.8	16.6	4.48
16×16	1250	1330	180	6.94
32×32	27.2s	30.0s	3.8s	7.17
64×64	7.8m	7.9m	1m	7.8
128×128	159.6m	160m	20.4m	7.82
256×256		very large time		

s denotes time in seconds and m denotes time in minutes

All other timings are in milliseconds

On Comparing the four set of timings we observe that FFT algorithms give the best timings. The timings have been reduced due to decrease in the number of computations. As compared to other two algorithms which require $2n^3$ multiplications and additions, FFT algorithms require $2n^2(n_1+n_2+1)$ multiplications and $2n^2(n_1+n_2-2)$ additions (where n_1 and n_2 are factors of n). As the size increases the speedup factor improves because communication time starts becoming insignificant compared to calculation time. The algorithm using Horner's rule has the best speedup factor but it also has the worst timings.

We have also implemented three algorithms for identification of alphabet. On comparing the timings for matching different alphabet we observe that timings are either 64 micro seconds or 128 micro seconds (i.e. 1 or 2 counts of internal clock). As we can not measure timings shorter than 64 micro seconds, therefore time comparison of these algorithms is not the suitable criterion for performance evaluation. Therefore it would be better to compare the average number of matchings required for identification.

Average Number of comparisons required in method 1 = 2.77 comparisons

Average Number of comparisons required in method 2 = 3.01 comparisons

Average Number of comparisons required in method 3 = 3.08 comparisons

6 2 SCOPE FOR FUTURE WORK

In the introduction of this thesis we started with some questions about the ability of computer in identification problem. These questions were

- (1) Can computers be used for identification purposes?
- (2) Can a method/methods be devised to be used for differentiating between two images which are almost similar?
- (3) Is the above method dependent upon the type of image?
- (4) Can this method operate in real time conditions?
- (5) What are the applications in which this method can be used?
- (6) Whether the method is commercially viable?

Let's see whether we can answer some of them now. We have been using computers for image identification but in a very limited way i.e. we can not identify a person or a fingerprint which is not in the computer memory. Also we have to match each image with all the images present in the memory. As the methods we have devised are also dependent on matching from memory therefore that limitation remains. As ultimately some form of matching is necessary for identification therefore that limitation will remain but we can reduce the dependence on memory in one way. If we can identify some property of image which is similar in one type of image (for ex. all fingerprints have vortex like structure) then we will have to match from a lesser number of images. What we should try is to classify the images in different groups. These groups can be formed on the basis of Galois transform also. It may be that all faces have some Galois coefficient/some transform which takes only so and so value.

We are at advantage when we are using Galois coefficients for identification of almost similar images because in almost similar images also the Galois coefficients are entirely different. The method we have developed is not dependent on type of image (if it's coefficients are stored in memory) but once we are able to classify the images then it would

be better to have different methods for different type of images

The method that we have developed takes about 64 to 128 micro seconds for comparison and about 10 msec to calculate coefficients. Therefore if we are able to reduce the time taken in calculation by using more and more nodes then this method could operate in real time. Thus we have to develop parallel programs which use more and more processors in efficient 3-dimensional networks.

As the purpose of method developed is to identify images therefore any application which requires image identification can use above method. Special Application can be in those areas where identification of two almost similar images is needed like fingerprint matching.

APPENDIX A

				A				
0	0	0	1	0	0	0	0	
0	0	1	0	1	0	0	0	
0	1	0	0	0	1	0	0	
1	1	1	1	1	1	1	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	1	0	
0	0	0	0	0	0	0	0	

D							
1	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

	G							
0	1	1	1	1	1	0	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	
1	0	0	1	1	1	1	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	1	0	
0	1	1	1	1	1	0	0	
0	0	0	0	0	0	0	0	

				J				
0	0	0	1	1	1	0	0	
0	0	0	0	1	0	0	0	
0	0	0	0	1	0	0	0	
0	0	0	0	1	0	0	0	
1	0	0	0	1	0	0	0	
1	0	0	0	1	0	0	0	
0	1	1	1	0	0	0	0	
0	0	0	0	0	0	0	0	

M								
1	0	0	0	0	0	1	0	
1	1	0	0	0	1	1	0	
1	0	1	0	1	0	1	0	
1	0	0	1	0	0	1	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	1	0	
0	0	0	0	0	0	0	0	

B							
1	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	1	1	1	1	1	0	0
1	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

E								
1	1	1	1	1	1	1	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	0	
0	0	0	0	0	0	0	0	

$$\begin{array}{cccccccc} & & & & & & & H \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

	K							
1	0	0	0	0	1	0	0	
1	0	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	
1	1	1	0	0	0	0	0	
1	0	0	1	0	0	0	0	
1	0	0	0	1	0	0	0	
1	0	0	0	0	1	0	0	
0	0	0	0	0	0	0	0	

	N							
1	0	0	0	0	0	1	0	
1	1	0	0	0	0	1	0	
1	0	1	0	0	0	1	0	
1	0	0	1	0	0	1	0	
1	0	0	0	1	0	1	0	
1	0	0	0	0	1	1	0	
1	0	0	0	0	0	1	0	
0	0	0	0	0	0	0	0	

	C							
0	1	1	1	1	1	0	0	
1	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	1	0	
0	1	1	1	1	1	0	0	
0	0	0	0	0	0	0	0	

				F				
1	1	1	1	1	1	1	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	

				I			
0	0	1	1	1	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0

								L
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0

$$\begin{array}{cccccccc} & & & & 0 & & & \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

P
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

S
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

V
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

Y
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

b
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

e
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 1 0
1 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0

Q
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 1 0 1 0
1 0 0 0 0 1 0 0
0 1 1 1 1 0 1 0
0 0 0 0 0 0 0 0

T
1 1 1 1 1 1 1 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

W
1 0 0 0 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
0 1 1 0 1 1 0 0
0 0 0 0 0 0 0 0

Z
1 1 1 1 1 1 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0

c
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

f
0 0 0 1 1 1 0 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

R
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 0 0
1 0 0 0 1 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

U
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

X
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 1 0 0 0 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

a
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0

d
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

g
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1
1 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0

h
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

k
1 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0
1 1 1 0 0 0 0 0
1 0 0 1 0 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0

n
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 0
0 1 0 1 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0

q
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0

t
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 1 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0

w
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
0 1 1 0 1 1 0 0
0 0 0 0 0 0 0 0

i
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0

l
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

o
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

r
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0

u
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 0 0 1 0 0
0 1 1 1 1 0 1 0
0 0 0 0 0 0 0 0

x
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 1 0 0 0
0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0

j
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0

m
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
0 0 0 0 0 0 0 0

p
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0

s
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
1 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

v
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0

y
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0

z
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

2
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0

5
1 1 1 1 1 1 1 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

8
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

0
0 1 1 1 1 1 0 0
1 0 0 0 0 1 1 0
1 0 0 0 1 0 1 0
1 0 0 1 0 0 1 0
1 0 1 0 0 0 1 0
1 1 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

3
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0

6
0 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0

9
0 1 1 1 1 1 0 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0

1
0 0 0 1 0 0 0 0
0 0 1 1 0 0 0 0
0 1 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0

4
0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0
0 0 1 0 1 0 0 0
0 1 0 0 1 0 0 0
1 1 1 1 1 1 1 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0

7
0 1 1 1 1 1 1 0
1 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0

APPENDIX B

a

0	0	0	0	0	0	0	0
1	0	4	7	4	6	6	7
7	5	1	3	3	4	0	1
6	2	7	1	0	5	1	5
3	3	5	4	3	2	1	7
4	3	2	6	1	1	2	0
2	2	6	5	4	3	2	1
5	5	3	2	1	7	6	5

b

1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
7	6	5	2	4	4	5	0
6	4	7	2	2	3	0	7
5	3	5	6	0	3	6	3
4	7	5	6	0	3	6	3
3	2	7	2	2	3	0	7
2	5	5	2	4	4	5	0

c

0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
7	3	0	3	6	3	5	5
6	5	5	0	2	5	2	4
1	6	6	6	7	0	4	7
4	2	2	2	3	0	7	3
1	7	7	0	4	7	4	6
1	4	0	4	7	4	6	6

d

0	1	6	4	2	7	5	3
0	0	0	0	0	0	0	0
2	0	6	6	7	0	4	7
3	0	0	4	7	4	6	6
5	7	1	3	3	4	0	1
5	0	7	0	4	7	4	6
3	4	2	6	1	1	2	0
2	6	7	1	0	5	1	5

e

0	0	0	0	0	0	0	0
1	1	7	6	5	4	3	2
7	0	4	4	5	0	2	5
6	0	0	7	3	7	2	2
1	2	2	5	2	4	4	5
4	0	6	0	3	6	3	5
1	5	3	6	3	5	5	6
1	3	7	3	7	2	2	3

i

0	1	3	5	7	2	4	6
0	0	5	2	1	3	1	1
0	5	2	3	7	2	2	5
0	2	3	3	3	5	2	6
0	7	1	1	1	3	7	4
0	3	2	5	3	5	4	5
0	4	1	2	6	1	1	4
0	6	5	1	6	1	7	1

j

0	0	0	0	0	0	0	0
1	1	7	6	5	4	3	2
6	1	4	0	1	4	1	3
4	1	7	7	1	0	5	1
2	5	5	2	4	4	5	0
7	1	0	6	2	6	1	1
5	3	5	6	0	3	6	3
3	2	7	2	2	3	0	7

k

1	1	0	0	0	0	0	0
1	0	3	5	5	2	3	2
7	3	5	3	4	5	2	4
6	5	2	2	3	5	7	7
5	7	5	7	5	2	1	2
4	2	2	3	6	3	6	5
3	4	4	5	5	3	3	1
2	6	3	2	1	6	3	2

l

0	1	3	5	7	2	4	6
0	1	3	5	7	2	4	6
0	7	2	4	6	1	3	5
0	6	1	3	5	7	2	4
0	5	7	2	4	6	1	3
0	4	6	1	3	5	7	2
0	3	5	7	2	4	6	1
0	2	4	6	1	3	5	7

m

0	0	0	0	0	0	0	0
0	0	1	1	6	1	7	4
4	4	6	4	4	1	7	7
7	7	1	4	6	7	6	7
6	6	1	6	6	3	2	2
6	6	6	1	4	7	6	4
7	7	7	2	5	1	7	5
4	4	5	1	3	4	3	4

q

0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
3	2	2	0	6	2	6	1
5	3	3	3	4	0	1	4
7	7	7	6	5	4	3	2
2	5	0	5	1	5	7	7
4	4	7	6	5	4	3	2
6	6	7	6	5	4	3	2

r

0	0	0	0	0	0	0	0
1	0	5	2	7	3	4	6
2	6	1	4	4	6	5	1
3	4	4	1	2	7	1	7
4	0	5	7	1	5	1	0
5	7	6	7	1	1	6	3
6	0	4	3	0	3	1	1
7	0	2	2	1	6	0	1

s

0	0	0	0	0	0	0	0
1	1	7	6	5	4	3	2
3	5	5	6	0	3	6	3
5	2	5	2	4	4	5	0
7	7	5	4	3	2	1	7
2	3	7	2	2	3	0	7
4	4	6	5	4	3	2	1
6	6	3	2	1	7	6	5

t

0	1	3	5	7	2	4	6
0	1	0	0	7	0	4	6
0	2	4	4	2	2	5	1
0	3	3	7	2	7	1	3
0	7	3	0	7	4	3	6
0	5	6	5	1	6	5	3
0	4	0	6	7	2	4	2
0	6	7	5	5	0	4	6

u

0	0	0	0	0	0	0	0
0	1	1	1	3	1	2	5
3	6	4	7	0	3	3	5
5	4	5	7	5	6	2	0
3	2	3	5	3	4	7	0
2	7	4	2	3	6	0	2
2	5	3	6	0	2	2	4
5	3	7	5	6	2	0	5

f
0 1 7 6 1 4 1 1
0 0 3 5 0 2 0 0
0 5 4 3 0 6 1 0
0 2 4 7 1 5 0 0
0 7 6 1 0 6 4 3
0 3 2 7 0 6 0 1
0 4 1 7 2 7 0 6
0 6 4 4 7 1 5 0

g
0 0 0 0 0 0 0 0
0 0 7 6 5 4 3 2
5 1 1 5 7 7 1 0
2 1 6 1 1 2 0 6
3 3 6 5 4 3 2 1
3 1 3 4 0 1 4 1
2 2 3 2 1 7 6 5
5 5 5 4 3 2 1 7

h
1 1 0 0 0 0 0 0
1 1 7 6 5 4 3 2
7 6 0 5 1 5 7 7
6 4 2 0 6 2 6 1
5 3 7 4 6 6 7 0
4 7 3 3 4 0 1 4
3 2 6 7 0 4 7 4
2 5 4 6 6 7 0 4

n
0 0 0 0 0 0 0 0
1 0 0 0 2 0 5 3
2 7 4 7 4 0 6 3
3 6 0 7 4 6 5 7
4 6 5 4 4 2 3 3
5 4 4 0 2 6 6 7
6 7 6 5 2 3 6 2
7 4 3 2 5 7 5 7

o
0 0 0 0 0 0 0 0
1 0 6 4 2 7 5 3
7 1 2 2 3 0 7 3
6 1 0 3 6 3 5 5
1 5 2 0 6 2 6 1
4 1 5 0 2 5 2 4
1 3 0 5 1 5 7 7
1 2 3 3 4 0 1 4

p
0 0 0 0 0 0 0 0
0 1 6 4 2 7 5 3
1 2 0 7 3 7 2 2
1 3 6 0 3 6 3 5
1 7 2 2 3 0 7 3
1 5 4 4 5 0 2 5
1 4 5 0 2 5 2 4
1 6 0 3 6 3 5 5

v
0 0 0 0 0 0 0 0
0 1 4 7 4 6 6 7
7 6 0 1 7 4 1 6
6 4 7 0 1 1 4 6
6 2 7 7 1 1 7 6
4 7 1 6 7 0 4 1
7 5 4 1 7 4 1 4
4 3 1 6 6 6 4 1

w
0 0 0 0 0 0 0 0
0 1 4 7 4 6 6 7
3 7 0 4 6 0 4 2
5 6 0 0 7 7 3 4
3 1 5 7 4 2 2 6
2 4 6 0 5 0 7 6
2 1 4 5 7 3 6 5
5 1 3 2 3 6 4 7

x
0 0 0 0 0 0 0 0
0 1 3 5 7 2 4 6
0 4 1 3 4 6 2 4
0 7 4 1 3 5 7 7
0 3 6 4 3 5 3 2
0 6 2 7 6 1 6 5
0 2 6 3 5 7 2 2
0 5 2 4 5 7 3 5

y
0 0 0 0 0 0 0 0
0 1 6 4 2 7 5 3
7 2 7 7 1 0 5 1
6 3 0 6 2 6 1 1
6 7 4 6 6 7 0 4
4 5 4 0 1 4 1 3
7 4 7 4 6 6 7 0
4 6 6 7 0 4 7 4

z
0 0 0 0 0 0 0 0
0 1 6 4 5 7 3 2
0 5 7 6 5 5 3 5
0 2 2 6 5 4 2 2
0 7 3 1 6 2 6 7
0 3 7 3 3 4 3 2
0 4 1 5 4 2 7 7
0 6 3 5 4 1 6 4

0
0 1 6 4 5 7 3 2
1 0 6 4 2 7 5 3
6 6 6 5 6 3 0 4
4 4 5 4 0 2 7 4
5 2 6 0 3 6 3 5
7 7 3 2 6 7 7 0
3 5 0 7 3 7 2 2
2 3 4 4 5 0 2 5

3
0 1 4 7 4 6 6 7
0 1 4 7 4 6 6 7
7 0 2 0 6 2 6 1
6 0 3 3 4 0 1 4
2 4 7 0 4 7 4 6
4 0 0 5 1 5 7 7
5 6 0 4 7 4 6 6
3 7 6 6 7 0 4 7

1
0 1 3 5 7 2 4 6
0 1 7 6 1 4 1 1
0 4 1 0 1 5 2 1
0 7 2 1 3 0 1 1
0 6 6 1 6 6 7 0
0 6 0 3 1 1 1 5
0 7 1 7 0 7 7 4
0 4 4 4 6 1 0 4

4
0 1 4 7 3 6 2 5
1 0 1 1 4 1 6 7
4 1 2 6 4 7 2 4
7 1 6 3 3 4 7 7
3 4 4 3 1 6 0 0
6 1 7 4 6 5 6 5
2 6 2 7 0 6 1 0
5 7 4 7 0 5 0 1

2
0 1 4 7 4 6 6 7
1 1 3 5 3 2 2 5
1 3 3 1 2 4 0 3
1 5 7 5 0 1 5 3
1 3 7 0 1 3 1 4
1 2 1 6 2 2 5 0
1 2 0 2 6 4 1 1
1 5 5 6 1 0 7 1

5
1 1 7 6 5 4 3 2
1 1 7 6 5 4 3 2
0 4 5 2 4 4 5 0
0 7 7 2 2 3 0 7
1 2 5 6 0 3 6 3
0 6 5 6 0 3 6 3
1 5 7 2 2 3 0 7
1 3 5 2 4 4 5 0

6
0 0 6 4 3 7 2 5
1 1 6 4 3 7 2 5
4 0 5 2 4 4 5 0
7 0 7 2 2 3 0 7
4 7 5 6 0 3 6 3
6 0 5 6 0 3 6 3
6 4 7 2 2 3 0 7
7 6 5 2 4 4 5 0

9
0 1 4 7 4 6 6 7
0 0 4 7 6 6 7 4
6 1 7 1 5 5 6 2
4 1 2 6 4 1 3 2
3 4 5 6 1 3 3 2
7 1 1 3 5 4 3 7
2 6 7 2 5 3 1 2
5 7 5 2 5 4 3 1

C
0 1 4 7 4 6 6 7
1 1 0 0 0 0 0 0
4 0 4 6 6 7 0 4
7 0 6 7 0 4 7 4
4 4 1 7 6 5 4 3
6 0 7 4 6 6 7 0
6 6 4 3 2 1 7 6
7 7 2 1 7 6 5 4

F
1 1 7 6 5 4 3 2
1 1 3 5 1 2 1 1
7 7 4 5 2 7 6 1
6 6 6 7 4 2 1 3
5 5 1 2 6 4 3 5
4 4 3 4 1 6 5 7
3 3 5 6 3 1 7 2
2 2 7 1 5 3 2 4

I
0 1 6 4 5 7 3 2
0 1 3 5 7 2 4 6
0 7 7 6 0 0 5 3
0 6 0 6 2 4 5 0
0 5 4 7 5 7 6 7
0 4 7 0 2 4 0 3
0 3 4 4 4 6 3 7
0 2 6 7 4 6 6 2

L
1 1 0 0 0 0 0 0
1 1 7 6 5 4 3 2
7 7 5 4 3 2 1 7
6 6 3 2 1 7 6 5
5 5 1 7 6 5 4 3
4 4 6 5 4 3 2 1
3 3 4 3 2 1 7 6
2 2 2 1 7 6 5 4

7
0 0 7 6 5 4 3 2
1 1 2 3 6 5 7 4
1 5 4 1 1 0 2 1
1 2 0 7 3 1 1 1
1 7 3 0 4 7 5 5
1 3 1 0 1 6 1 5
1 4 0 4 3 2 6 3
1 6 6 5 2 0 2 7

A
0 1 3 5 7 2 4 6
0 0 0 0 0 0 0 0
4 3 5 0 1 7 2 4
7 5 6 2 3 0 7 1
6 7 2 1 3 2 6 0
6 2 0 4 6 3 1 5
7 4 1 5 0 5 2 7
4 6 3 3 4 1 0 5

D
1 0 4 7 4 6 6 7
1 0 6 4 2 7 5 3
7 0 0 1 4 1 3 3
6 0 1 0 5 1 5 7
5 0 4 5 0 2 5 2
4 0 1 1 2 0 6 2
3 0 3 5 5 6 0 3
2 0 3 7 2 2 3 0

G
0 1 4 7 4 6 6 7
1 0 7 6 1 4 1 1
4 4 3 5 4 6 6 4
7 7 4 5 4 2 7 7
4 1 3 7 1 2 0 5
6 6 3 7 6 2 6 7
6 1 4 5 3 2 1 0
7 1 3 5 0 6 2 1

J
0 1 7 6 1 4 1 1
0 1 3 5 5 2 3 2
2 3 2 2 4 6 6 0
3 5 4 3 4 3 0 7
5 0 4 0 5 5 3 3
5 2 5 7 0 5 6 7
3 0 0 3 2 6 3 2
2 0 2 7 5 0 5 2

M
1 0 6 4 2 7 5 3
1 1 7 6 5 4 3 2
7 3 4 4 6 7 6 4
6 5 6 7 4 7 7 4
5 7 1 0 5 2 3 0
4 2 6 4 6 6 7 7
3 4 0 5 0 1 3 2
2 6 3 1 5 0 0 2

8
0 1 4 7 4 6 6 7
0 1 4 7 4 6 6 7
1 5 2 0 6 2 6 1
1 2 3 3 4 0 1 4
6 3 7 0 4 7 4 6
1 3 0 5 1 5 7 7
7 2 0 4 7 4 6 6
4 5 6 6 7 0 4 7

B
1 0 4 7 4 6 6 7
1 0 4 7 4 6 6 7
7 0 2 0 6 2 6 1
6 0 3 3 4 0 1 4
5 0 7 0 4 7 4 6
4 0 0 5 1 5 7 7
3 0 0 4 7 4 6 6
2 0 6 6 7 0 4 7

E
1 1 7 6 5 4 3 2
1 1 2 3 7 5 4 6
7 7 2 2 7 3 2 5
6 6 5 3 3 3 2 6
5 5 0 3 0 2 1 6
4 4 5 2 3 5 4 5
3 3 2 5 7 0 0 1
2 2 3 0 1 5 4 0

H
1 0 6 4 2 7 5 3
1 1 7 6 5 4 3 2
7 3 3 7 2 2 3 0
6 5 3 5 5 6 0 3
5 7 0 2 5 2 4 4
4 2 4 5 0 2 5 2
3 4 5 5 6 0 3 6
2 6 3 0 7 3 7 2

K
1 0 5 2 6 3 7 4
1 1 6 4 3 7 2 5
7 3 5 3 4 5 2 4
6 5 2 2 3 5 7 7
5 7 5 7 5 2 1 2
4 2 2 3 6 3 6 5
3 4 4 5 5 3 3 1
2 6 3 2 1 6 3 2

N
1 0 6 4 2 7 5 3
1 1 7 6 5 4 3 2
7 4 1 1 2 0 6 6
6 7 0 1 4 1 4 3
5 4 5 2 4 3 5 0
4 6 1 0 7 1 5 7
3 6 5 2 0 3 6 3
2 7 5 2 2 3 0 7

O
0 1 4 7 4 6 6 7
1 0 6 4 2 7 5 3
4 6 0 1 4 1 3 3
7 4 1 0 5 1 5 7
4 2 4 5 0 2 5 2
6 7 1 1 2 0 6 2
6 5 3 5 5 6 0 3
7 3 3 7 2 2 3 0

R
1 0 4 7 4 6 6 7
1 0 1 1 4 1 6 7
7 0 5 4 3 4 6 2
6 0 7 2 4 7 3 5
5 0 3 6 5 3 0 4
4 0 6 6 5 3 2 7
3 0 7 2 6 2 3 0
2 0 5 5 0 4 7 2

U
1 0 6 4 2 7 5 3
0 1 4 7 4 6 6 7
4 6 0 1 4 1 3 3
7 4 1 0 5 1 5 7
4 2 4 5 0 2 5 2
6 7 1 1 2 0 6 2
6 5 3 5 5 6 0 3
7 3 3 7 2 2 3 0

X
1 0 6 4 2 7 5 3
0 1 3 5 7 2 4 6
6 3 6 0 2 0 4 4
5 0 4 7 0 7 3 1
2 7 2 7 2 6 4 7
7 2 0 0 6 7 5 6
5 4 4 7 4 5 5 6
3 6 4 3 7 6 6 3

P
1 0 4 7 4 6 6 7
1 1 0 0 0 0 0 0
7 1 7 0 4 7 4 6
6 1 6 6 7 0 4 7
5 4 4 0 1 4 1 3
4 1 0 4 7 4 6 6
3 6 0 6 2 6 1 1
2 7 7 7 1 0 5 1

S
0 1 4 7 4 6 6 7
1 1 7 6 5 4 3 2
3 0 3 5 5 6 0 3
5 0 4 5 0 2 5 2
5 4 3 7 2 2 3 0
2 0 3 7 2 2 3 0
3 6 4 5 0 2 5 2
2 7 3 5 5 6 0 3

V
1 0 6 4 2 7 5 3
0 1 4 7 4 6 6 7
5 6 6 3 3 6 7 5
2 4 4 4 6 5 2 5
2 2 4 6 6 5 1 5
3 7 2 7 3 7 2 4
5 5 7 3 3 6 7 1
3 3 2 7 1 4 2 4

Y
1 0 6 4 2 7 5 3
1 0 2 3 1 5 1 1
0 4 7 7 3 5 6 3
0 7 2 6 4 6 5 5
0 6 7 0 0 0 0 3
0 6 4 3 2 4 2 7
0 7 0 0 2 4 0 0
0 4 0 6 0 0 5 0

Q
0 1 4 7 4 6 6 7
1 1 7 6 7 4 4 6
4 7 1 6 2 7 5 1
7 6 6 1 2 4 1 3
4 7 2 2 7 1 6 4
6 4 7 4 1 1 5 3
6 4 5 1 6 5 4 7
7 6 1 3 4 3 7 6

T
1 1 7 6 5 4 3 2
1 1 7 6 5 4 3 2
0 7 2 4 6 1 3 5
0 6 1 3 5 7 2 4
0 5 7 2 4 6 1 3
0 4 6 1 3 5 7 2
0 3 5 7 2 4 6 1
0 2 4 6 1 3 5 7

W
1 0 6 4 2 7 5 3
0 1 4 7 4 6 6 7
4 4 2 3 7 0 0 6
7 7 0 3 0 5 4 6
4 4 6 4 4 1 7 7
6 6 2 0 7 5 4 0
6 6 6 1 4 7 6 4
7 7 1 4 6 7 6 7

Z
1 1 7 6 5 4 3 2
0 1 4 7 4 6 6 7
6 7 0 3 1 6 4 2
4 6 4 0 7 5 3 1
2 5 3 1 0 4 2 7
7 4 2 7 5 0 1 6
5 3 1 6 4 2 0 5
3 2 7 5 3 1 6 0

REFERENCES

INMOS Ltd , *Transputer Development system*, Prentice—Hall Inc, 1988

INMOS Ltd , *Transputer Reference Manual*, Prentice—Hall Inc, 1988

INMOS Ltd , *Occam 2 Reference Manual*, Prentice—Hall Inc, 1988

Pountain D and May D, "*A Tutorial introduction to Occam Programming*", BSP Professional Books, 1987

Varghese George, "*Galois switching functions Algebraic structure and applications*", Ph D thesis, Electrical Dept I I T Kanpur, 1990

Blahut R E, "*Fast algorithms for digital signal processing*", Addison—Wesley, 1985

Subbarao M K V, "*Transputer implementation of systolic arrays*", M Tech Thesis, Electrical Dept I I T Kanpur, 1992

Srinivas Galgali, "*Transputer Implementation of signal processing algorithms*", Tech Thesis, Electrical Dept I I T Kanpur, 1992

Quinn M J, "*Designing efficient algorithms for parallel computers*", McGraw—Hill International, 1987

Kung S Y, "*VLSI array processors*", Prentice Hall, New Jersey, 1988

Wexler John and Prior D, "*Solving problems with Transputes*", Microprocessors and systems, March 1989

A

115435